Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Investigación en comunicación gráfico visual el diseño de mensajes relativos a la limpieza y el valor del agua en Charadu (India).

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Nerea Fernández Fraga

Tutor: Santiago Escobar Román

Curso 2019-2020

Resumen

El siguiente documento detalla el proceso de desarrollo de una aplicación móvil basada en una aplicación educativa semejante a un juego para los niños y niñas de Charadu (India). Consta de un total de tres actividades interactivas con el fin de contribuir al conocimiento y la difusión de buenos usos y prácticas del agua.

Para su realización se utiliza *Unity3D*, en concreto las herramientas nativas 2D del programa, junto al lenguaje de programación *C#* bajo el IDE *Visual Studio Code*. Además del SDK de Android para la generación de APK's (gracias a la instalación del programa Android Studio).

Por otra parte, se ha hecho uso de *Assets* propios de Unity y del programa *Adobe Photoshop* para toda la cuestión del diseño del juego (frames, paneles, fuentes, fondos…).

Palabras clave: Ciclo del agua, Unity, C#, *Assets*…

Abstract

The following document details the development process of a mobile application based on an education application similar to a game for children of Charadu (India). It consists of three interactive activities in order to contribute to the knowledge and dissemination of good uses of water.

For its realization, *Unity3D* is used, specifically the 2D native tools, with the C# programming language under the Visual Studio Code IDE. Besides to the Android SDK for the generation of APK (thanks to the installation of the Android Studio program).

On the other hand, the Unity own Assets and the Adobe Photoshop program have been used for the whole question of game design (frames, panels, fonts, background…).

Keywords: Water cycle, Unity, C#, *Assets*…

Resum

El següent document detalla el procés de desenvolupament d'una aplicació mòbil basada en una aplicació educativa semblant a un joc per als xiquets i xiquetes de Charadu (Índia). Consta d'un total de tres activitats interactives a fi de contribuir al coneixement i la difusió de bons usos i pràctiques de l'aigua.

Per a la seua realització s'utiliza *Unity3D*, en concret les ferramentes natives 2D del programa, junt amb el llenguatge de programació *C#* davall l'IDE *Visual Studio Code*. A més del SDK d'Android per a la generació d'APK's (gràcies a la instal·lació del programa Android Studio).

D'altra banda, s'ha fet ús d'*Assets* propis d'Unity i del programa *Adobe Photoshop* per a tota la qüestió del disseny del joc (frames, panels, fonts, fondos…).

Palabras clave: Cicle de l'aigua, Unity, C#, *Assets*…

Tabla de contenidos

1. Introducción

India es un país mayoritariamente subdesarrollado, donde no todas las personas cuentan con los mismos recursos y donde, al mismo tiempo, éstos se alejan mucho de los que posee cualquier país desarrollado. El acceso a la educación sigue siendo un problema y una barrera clara para todos sus habitantes. Este hecho, sumado a la clara discriminación presente en el territorio, hace que India sea de los países con menos alfabetización del mundo.

Por otra parte, en una nación donde abunda la desigualdad, de todos los tipos, y el analfabetismo, la tecnología es uno de los campos en los que destaca a nivel mundial. En busca de un aumento de esa carente alfabetización, la India ha impulsado y fomentado el uso de las TIC dentro de la educación y han optado por soluciones tecnológicas para otorgar una enseñanza de más calidad, mediante proyectos en los que se han entregado *tablets* a estudiantes de forma gratuita [8], o incluso en los que se han instalado ordenadores en las zonas más marginales del país.

1.1 Motivación

Actualmente, el uso de las Tecnologías de la Información ha aumentado considerablemente a nivel mundial, ya que brindan un conjunto de servicios con el fin de beneficiar y ayudar a las personas. Su incorporación a la enseñanza nace como necesidad y busca el avance en el aprendizaje.

Tener la posibilidad de contribuir a esa mejora de la situación en el país hindú haciendo que niños y niñas aprendan mientras se entretienen y descubren cosas nuevas al mismo tiempo es la principal motivación de este trabajo.

Esta aplicación nace como colaboración con la profesora Marina Puyuelo Cazorla del Departamento de Expresión Gráfica Arquitectónica y adscrita a la Escuela Técnica Superior de Ingeniería de Diseño, que es la investigadora principal del proyecto *“Investigación en Comunicación Gráfico Visual para el diseño de mensajes relativos a la limpieza y el valor del agua. Aplicación en el pueblo de Charadu, India”* (del programa *ADSIDEO 2018 de cooperación al desarrollo de la UPV)*. Dicha colaboración consistía en el desarrollo de una aplicación móvil en la que, a partir de actividades interactivas, los estudiantes hindúes aprenden conceptos desconocidos sobre los buenos usos y prácticas del agua a la vez que juegan.

1.2 Objetivos

El objetivo principal de este proyecto ha sido crear una aplicación multiplataforma para dispositivos móviles, la cual contiene un total de tres minijuegos interactivos diferentes que el usuario tiene que completar en el menor tiempo posible y con los que se pretende adquirir conocimientos acerca de buenos usos del agua. Asimismo, hay que tener en cuenta el detalle de que debe ser de una complejidad sencilla para hacerla accesible a una gran cantidad de gente, teniendo en cuenta el hecho del analfabetismo de la población.

1.3 Estructura de la memoria

Este documento está compuesto de los siguientes diez apartados:

1. **Introducción:** se explica la motivación y los objetivos del proyecto realizado.

2. **Estado del arte:** se analiza la situación del mercado al que pertenece el proyecto que se va a desarrollar.

3. **Especificación de requisitos:** se describirán las técnicas empleadas para el análisis y elicitación de requisitos y la metodología del desarrollo empleada.

4. **Diseño:** se detalla el proceso de definición previo en cuanto a aspectos más técnicos cercanos a la implementación con tal de obtener una representación de lo que se pretende construir.

5. **Tecnología y herramientas:** se enumeran todas las tecnologías y herramientas utilizadas a lo largo de todo el proyecto.

6. **Implementación:** se tratan aspectos considerados relevantes del desarrollo de la aplicación, así como los cambios producidos y los procedimientos de trabajo empleados.

7. **Pruebas:** se exponen aspectos considerados relevantes en la fase de pruebas de la aplicación.

8. **Conclusión:** se resume lo logrado en el proyecto, los problemas que se han presentado y mejoras que podrían llevarse a cabo en un futuro.

9. **Referencias:** se incluye la bibliografía de las fuentes utilizadas a lo largo del proyecto.

2. Estado del arte

En esta parte, tras un largo análisis, se estudia y documentan otras aplicaciones de funcionalidades similares a la del proyecto desarrollado que existan o hayan existido en el mercado.

Este tipo de aplicación pertenece al área de desarrollo de programas conocida como gamificación, una técnica de aprendizaje que consiste en incorporar la mecánica de los juegos al ámbito educativo.

2.1 Crítica

En primer lugar, se ha realizado una búsqueda en profundidad de aplicaciones de temática idéntica a la elaborada. Al ser tan específica, ésta no ha sido muy exitosa, pues actualmente no abundan las aplicaciones que sean juegos educativos para niños y niñas con el fin de enseñarles buenas prácticas sobre el agua.

No obstante, se pueden destacar dos: *Nature Bert Earth* y *El increíble chapuzón de Zoe.*

Por otro lado, al obtener tan pocos resultados anteriormente, se ha realizado una búsqueda más global dentro del sector de gamificación, con el fin de encontrar aplicaciones con el mismo objetivo y contexto que la desarrollada, pero sin tener que ser de la misma temática.

En esta segunda búsqueda la aplicación más popular, similar y utilizada en este sentido es *Knowre Math.*

2.1.1 Nature Bert Earth

Nature Bert Save the Earth : Juegos educativos para niños 4+

1tucan

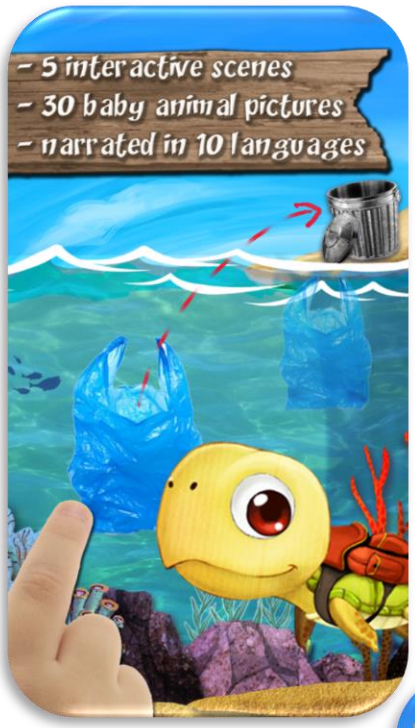
★★★★★ 1,0, 1 valoración

3,49 €

Esta aplicación pretende enseñar a los más pequeños acciones simples para proteger al planeta y conseguir salvar y ver felices a los animales que están en peligro en cada escena diferente [3].

Cerrar el grifo para que el caballo pueda tomar agua, ducharse rápido para que los patos puedan nadar, no tirar bolsas de plástico para que las tortugas no las coman… Las actividades diarias a las que se está acostumbrado se convierten en impactos negativos para el medio ambiente con los que se pretende concienciar a los niños y niñas de su papel en el cuidado del planeta.

La edad recomendada es entre uno y cinco años y su última actualización fue el 03 de agosto del 2015.



2.1.2 El increíble chapuzón de Zoe

El increíble chapuzón de Zoe 4+
European Union Apps
★★★★☆ 4,0, 5 valoraciones
Gratis

Esta aplicación ha sido desarrollada por la Dirección General de Medio Ambiente de la Comisión Europea. Pretende orientar a los niños y niñas sobre cómo pueden ayudar a salvar el medio ambiente, señalando en las diferentes escenas los errores en la conducta que se deben corregir para conseguirlo [5].

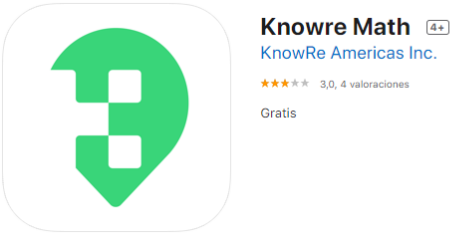
En forma de cuento interactivo se narra la historia de dos niños pequeños con Fred, una rana parlante muy lista que les sirve de guía, y gracias a la cual conocen las consecuencias de la contaminación del agua y aprenden cómo evitarla.

Desde el punto de vista práctico, ofrece muchos consejos para conservar el agua, concienciando de la importancia de ésta en la sociedad, el impacto de su contaminación, cómo hacer un uso responsable de ella para evitarlo y sus beneficios.

También contiene secciones más teóricas sobre el ciclo hidrológico y la depuración del agua y viene acompañado de notas para profesores en las que se proponen actividades que se puedan realizar en las aulas.

La edad recomendada es entre siete y once años y su última actualización fue el 07 de diciembre del 2012.

2.1.3 Knowre Math



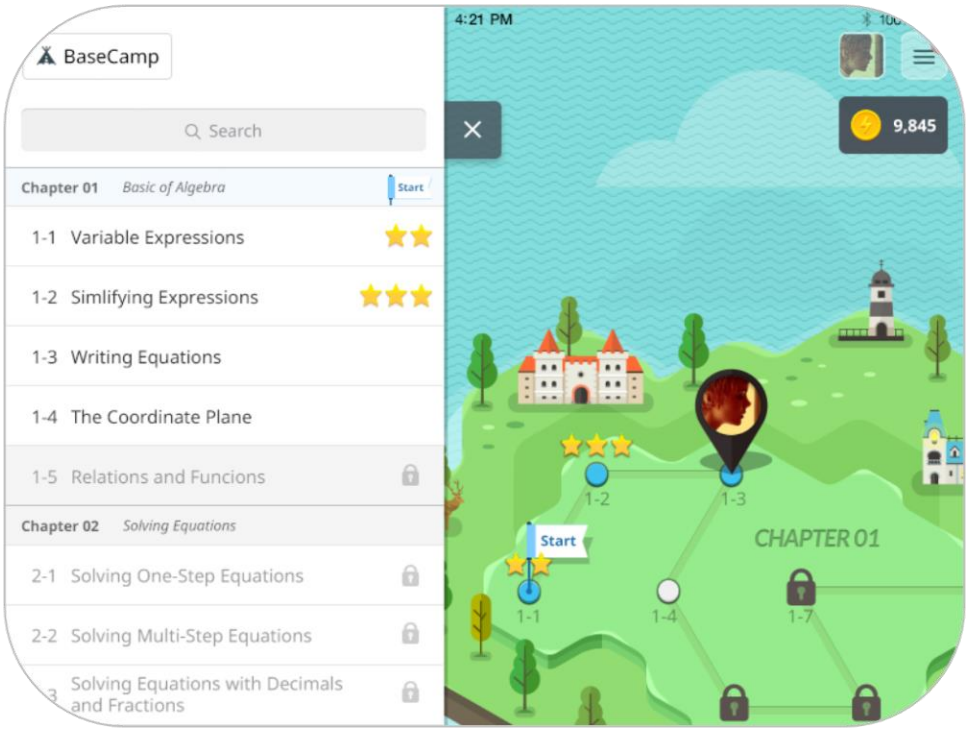
Esta aplicación nació de una academia en Seúl, Corea del Sur, y ha ido creciendo considerablemente hasta implantarse en treinta y cuatro ciudades y diecisiete estados de Estados Unidos y en Canadá [10].

Los usuarios van recorriendo toda una región en la que se va poniendo a prueba los conocimientos que van adquiriendo para la resolución de los problemas matemáticos que van saliendo en el transcurso, todos relacionados con la vida cotidiana.

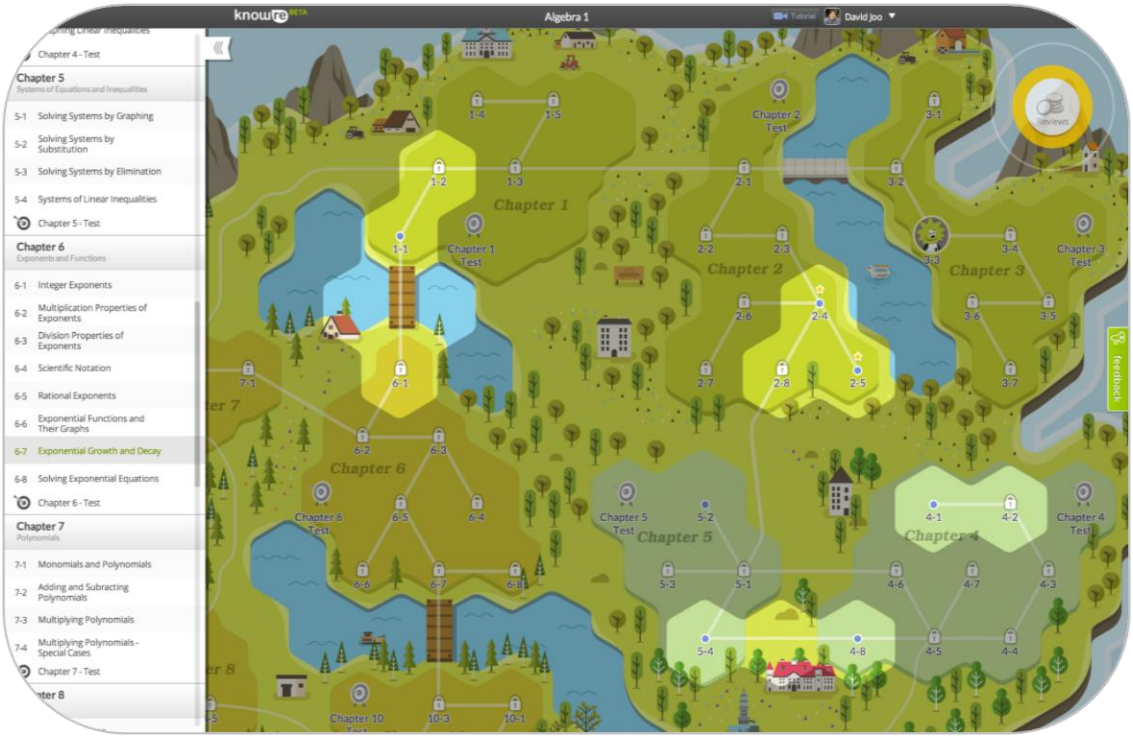
Además, al finalizar cada etapa del juego se realiza un test de repaso, donde se indican los apartados en los que se ha fallado.

Como peculiaridad presenta que en ningún momento se muestra la solución ni el procedimiento para llegar a ella, sino que se van lanzando preguntas más sencillas para que se pueda ir resolviendo parcialmente. Así, poco a poco y con mucha práctica se consigue la capacidad de resolver el problema por completo.

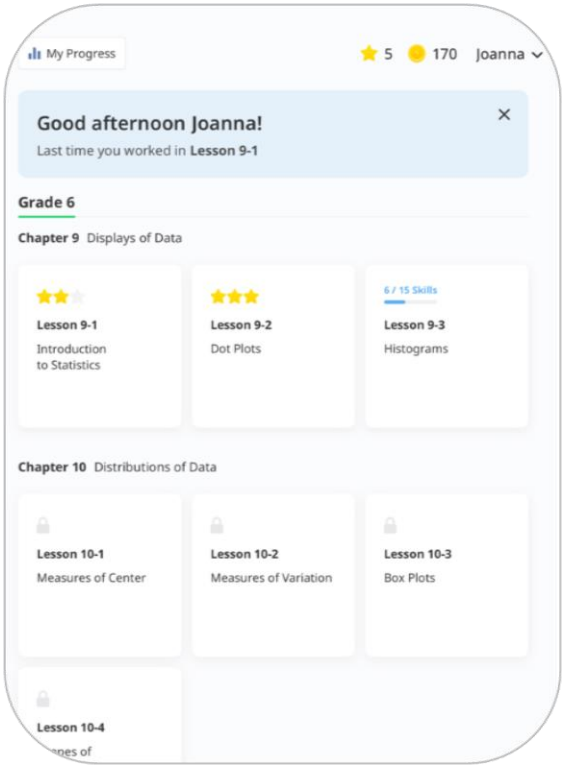
La edad recomendada es para mayores de cuatro años, sin embargo, va enfocada principalmente a estudiantes de secundaria y en la actualidad sigue en desarrollo.

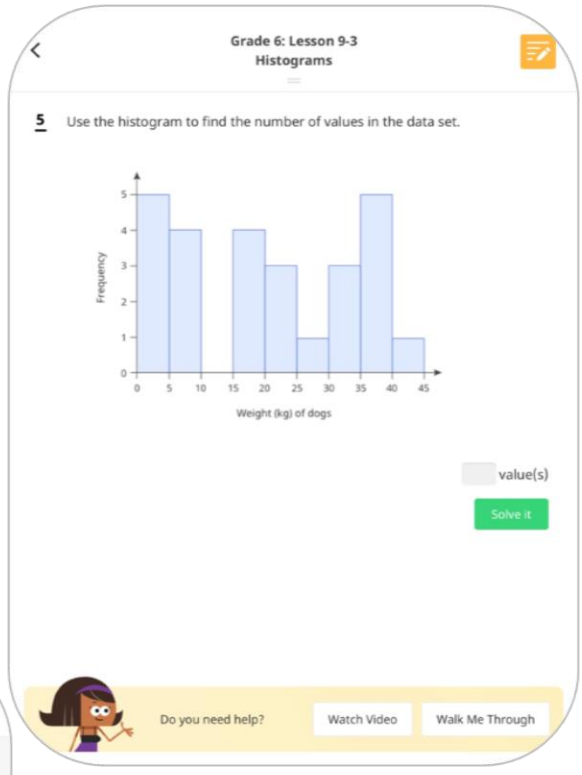


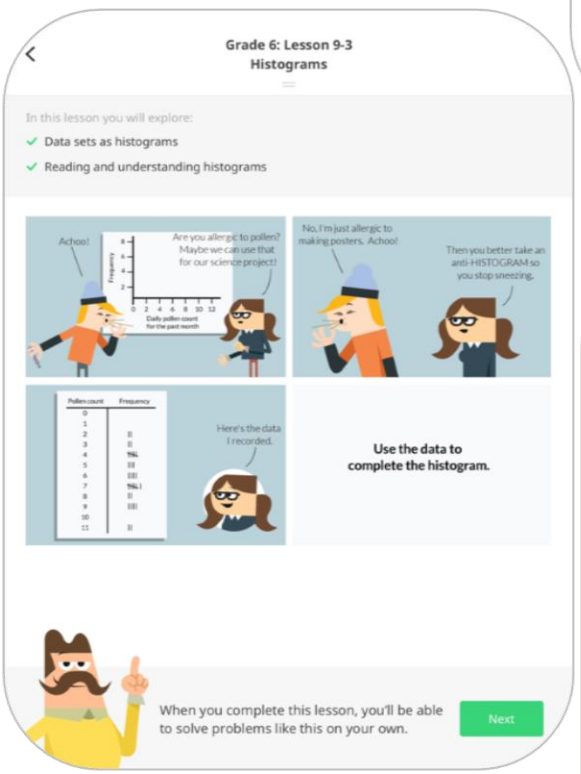
Página principal versión aplicación

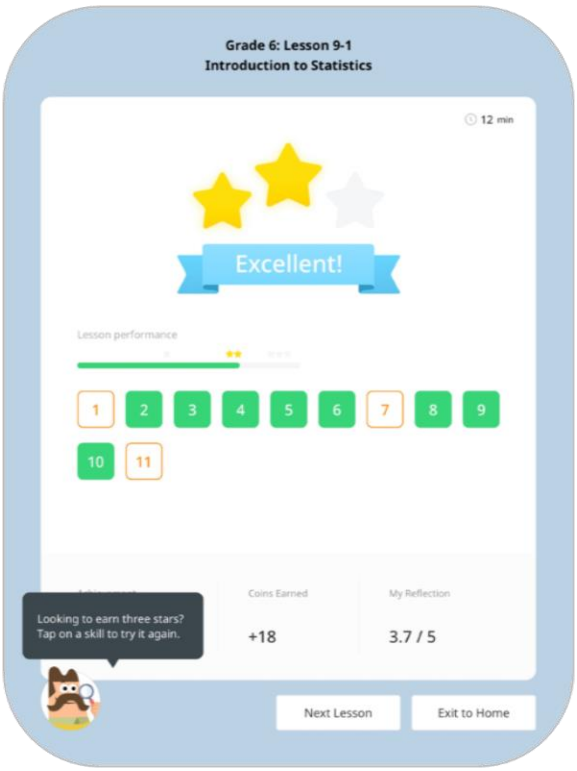


Página principal versión web


My Progress
5
170
Joanna
Good afternoon Joanna!
Last time you worked in Lesson 9-1
Grade 6
Chapter 9 Displays of Data
Lesson 9-1
Introduction to Statistics
Lesson 9-2
Dot Plots
6 / 15 Skills
Lesson 9-3
Histograms
Chapter 10 Distributions of Data
Lesson 10-1
Measures of Center
Lesson 10-2
Measures of Variation
Lesson 10-3
Box Plots
Lesson 10-4


Grade 6: Lesson 9-3
Histograms
5 Use the histogram to find the number of values in the data set.
Frequency
Weight (kg) of dogs
value(s)
Solve it
Do you need help?
Watch Video
Walk Me Through


Grade 6: Lesson 9-3
Histograms
In this lesson you will explore:
Data sets as histograms
Reading and understanding histograms
Achoo!
Are you allergic to pollen? Maybe we can use that for our science project!
No, I'm just allergic to making posters. Achoo!
Then you better take an anti-HISTOGRAM so you stop sneezing.
Here's the data I recorded.
Use the data to complete the histogram.
When you complete this lesson, you'll be able to solve problems like this on your own.
Next


Grade 6: Lesson 9-1
Introduction to Statistics
12 min
Excellent!
Lesson performance
1
2
3
4
5
6
7
8
9
10
11
Looking to earn three stars? Tap on a skill to try it again.
Coins Earned
+18
My Reflection
3.7 / 5
Next Lesson
Exit to Home

2.2 Propuesta

Como se ha podido observar en el apartado anterior y centrándonos en las aplicaciones de la misma temática, ninguna de ellas se encuentra disponible en Google Play (Android) y, en general, se han dejado de mantener, llegando a ser incompatibles para ciertos dispositivos de versiones de sistemas operativos posteriores. Por lo tanto, hay escasa competencia y se tiene una gran oportunidad en cuanto a mercado.

Tanto las aplicaciones mencionadas como la desarrollada coinciden en el mismo objetivo y es el que mediante juegos y dibujos los más pequeños aprendan mientras se divierten, por lo que todas buscan el aprendizaje basado en el entretenimiento, cumpliendo a la perfección el concepto de gamificación.

Nature Bert Earth, al ir dedicada a un sector infantil de menor edad, se ciñe a conceptos más básicos para hacerla sencilla. Además, trata aspectos más generales, es decir, les enseña a cómo proteger el medio ambiente con acciones beneficiosas tanto para el agua como para la electricidad.

Por lo tanto, en cuanto a conceptos teóricos se refiere, *El increíble chapuzón de Zoe* es la más similar con la desarrollada, las dos mantienen la misma línea, con conceptos más complejos y centrados en el uso del agua. Aunque la de este proyecto abarca más conceptos, lo que marca una diferencia y aporta novedad.

Respecto a *Knowre Math* coinciden en el uso de diferentes niveles a lo largo del juego, cada uno asociado a un temario diferente, con el que se pretende que el usuario ponga a prueba sus conocimientos obtenidos y del que se le proporciona un *feedback* al acabar en forma de puntuación.

Otro aspecto que se puede añadir para hacerla más novedosa es el concepto de competición, pues, aunque no haya un registro y control de los usuarios que permita la “rivalidad” entre ellos, el hecho de que haya un cronómetro que vaya contando el tiempo empleado en cada minijuego es un aliciente para incentivar al usuario en ese sentido.

3. Especificación de requisitos

A continuación, se van a enunciar las diferentes técnicas empleadas a la hora de la obtención de requisitos. Esta fase nos permite comprender y situarnos próximos al problema.

El análisis de requisitos trata de capturar y describir las características del producto software que se va a realizar. Puede parecer una tarea fácil, pero nada más lejos de la realidad. Aquí entra en juego el factor de la comunicación y en más de una ocasión se han producido situaciones en los que cliente e ingeniero/a difieren en la idea que se tenía del resultado final a causa de una falta de entendimiento.

En el caso de nuestra aplicación, esta fase era determinante, pues al ser un proyecto de colaboración con otro departamento, el cual carecía de conocimientos en la materia, desde el primer momento se diferenció el papel de cliente. Se tenía que entender muy bien qué se pedía y cómo se quería.

3.1 Técnicas de elicitación

En la especialidad de la ingeniería del software existen diversas técnicas para la elicitación de requisitos, de las cuales se han hecho uso de las siguientes:

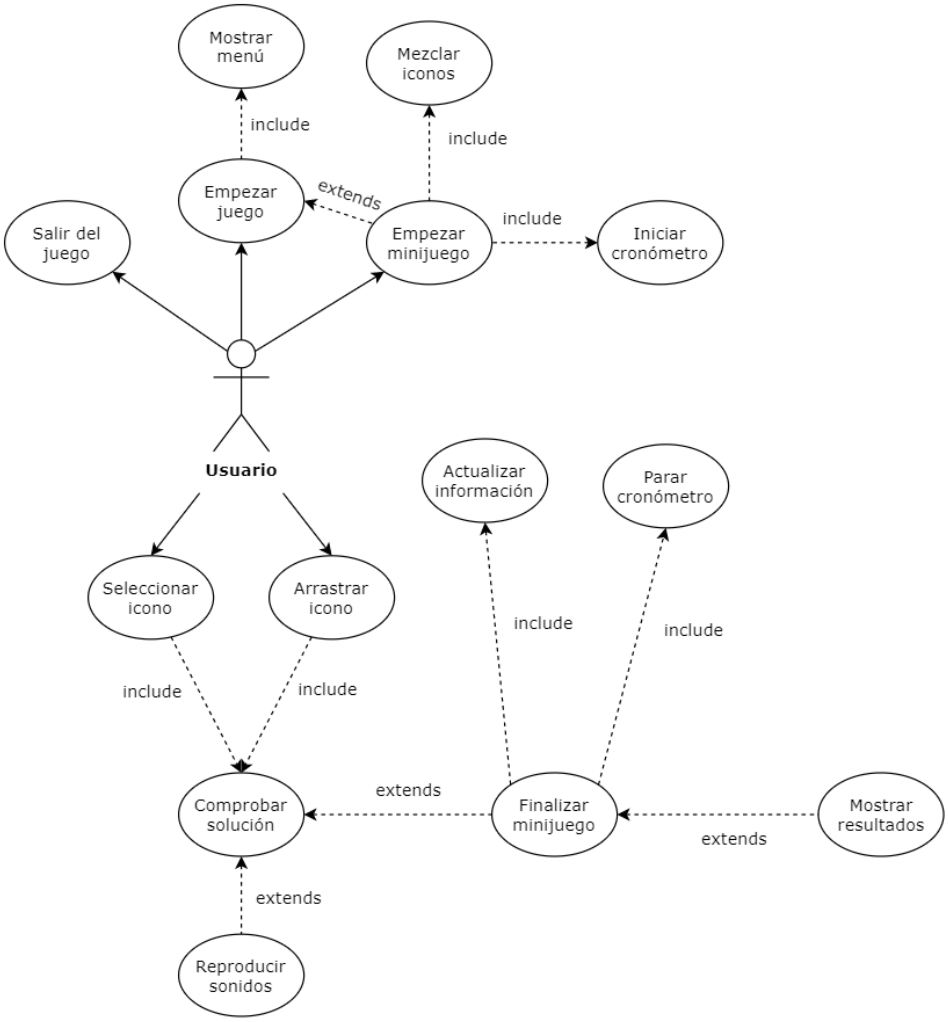
- **Entrevistas**: es una técnica tradicional usada muy frecuentemente. Han consistido en diferentes reuniones centradas principalmente en la descripción de las necesidades y en la resolución de cuestiones concretas por parte del cliente sobre el problema a resolver.

- **Prototipado:** es otra técnica muy utilizada y útil para que los *stakeholders* puedan comprender con más facilidad el sistema a desarrollar. Es un medio común entre la elicitación y la validación de requisitos. Se ha llevado a cabo mediante la realización de *mock-ups* de IU, referidos a las escenas de cada minijuego.

3.2 Requisitos

A continuación, se van a resumir los requisitos que debe cumplir la aplicación, clasificados en funcionales y no funcionales.

3.2.1 Requisitos funcionales



En el diagrama anterior aparecen reflejados todos los casos de uso que aparecen en la aplicación. Su comportamiento es típico y su nombre los describe a la perfección, sin embargo, hay algunos que tienen más particularidades y se van a detallar en profundidad:

Caso de uso	**Mostrar menú**
Precondición	Juego comenzado.
Postcondición	Mostrar menú con acceso a los niveles desbloqueados y detectar aquellos que han sido completados y aquellos que siguen bloqueados.

Caso de uso	Empezar minijuego
Precondición	El usuario debe de haber comenzado el juego en algún momento y el minijuego seleccionado tiene que estar desbloqueado.
Postcondición	Minijuego actual en transcurso.

Caso de uso	Finalizar minijuego
Precondición	La solución del usuario es correcta.
Postcondición	• Minijuego actual → completo • Minijuego siguiente → desbloqueado

Caso de uso	Actualizar información
Precondición	Minijuego empezado y la solución del usuario es correcta.
Postcondición	• Minijuego actual + 1. • Tiempo del minijuego actual guardado.

Caso de uso	Parar cronómetro
Precondición	Minijuego empezado y la solución del usuario es correcta.
Postcondición	Tiempo detenido en el instante que el usuario introduce la solución correcta.

Caso de uso	Mostrar resultados
Precondición	Juego completo (todos los minijuegos han sido superados)
Postcondición	Mostrar el tiempo empleado en resolver cada minijuego ordenados de menor a mayor.

3.2.2 Requisitos no funcionales

En la siguiente tabla se van a resumir los principales requisitos no funcionales exigidos para la aplicación. Se han clasificado por características siguiendo el modelo de calidad ISO 25010 y las principales categorías definidas en él.

USABILIDAD **Capacidad del producto para ser entendido, aprendido, usado y resultar atractivo para el usuario**	1. El usuario debe aprender a usar de manera rápida y sencilla la aplicación. 2. Interfaz lo más intuitiva posible. 3. Uso de poco texto. 4. Orientación *portrait* para toda la aplicación.
COMPATIBILIDAD **Capacidad de dos o más sistemas de intercambiar información y/o llevar a cabo funciones requeridas**	1. Aplicación multiplataforma. 2. Diseño *responsive* para garantizar la correcta visualización de la aplicación y adaptarse a diferentes resoluciones de pantalla. 3. Formato de audio universal.
EFICIENCIA **Desempeño relativo a la cantidad de recursos utilizados**	1. Tiempo de respuesta mínimo (menor a dos segundos). 2. No ocupar mucho espacio de almacenamiento.

3.3 Prototipado

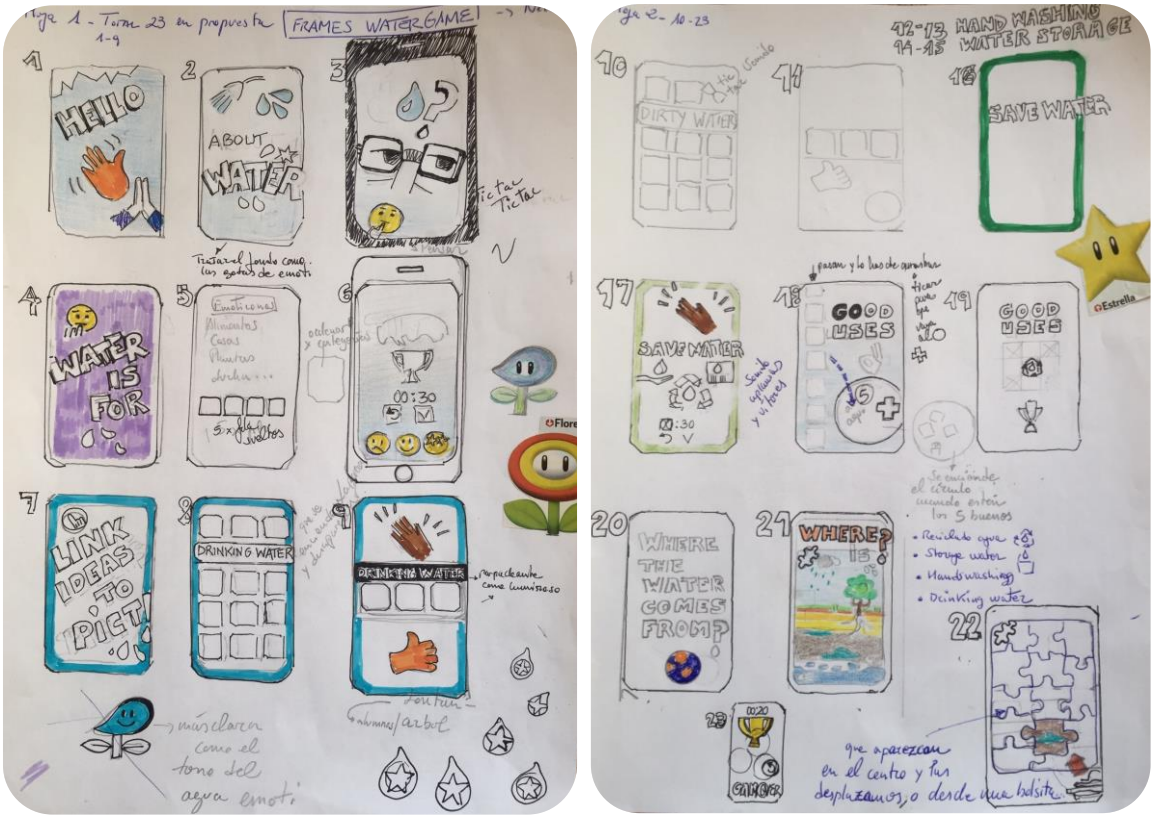
Esta técnica consiste en modelar el producto final, efectuando pruebas sobre determinados aspectos sin necesidad de que esté disponible en ese momento. Por lo tanto, un prototipo es un modelo (representación o simulación) fácilmente ampliable y/o modificable del sistema, incluyendo su interfaz y algunas determinadas funcionalidades de entradas y salidas.

Existen muchísimas opciones, herramientas y características para elaborarlos, teniendo así un amplio abanico de posibilidades. No obstante, se ha optado por la realización de prototipos de baja fidelidad *(Low-Fidelity prototyping*), implementados con lápiz y papel.

Este tipo de prototipado consiste en dibujar las interfaces que se van a desarrollar, en este caso con algunos detalles estéticos puntuales, emulando las interacciones del sistema. Son prototipos simples, pero muy versátiles, por lo que su gran ventaja es la velocidad y flexibilidad a la hora de realizarlos.

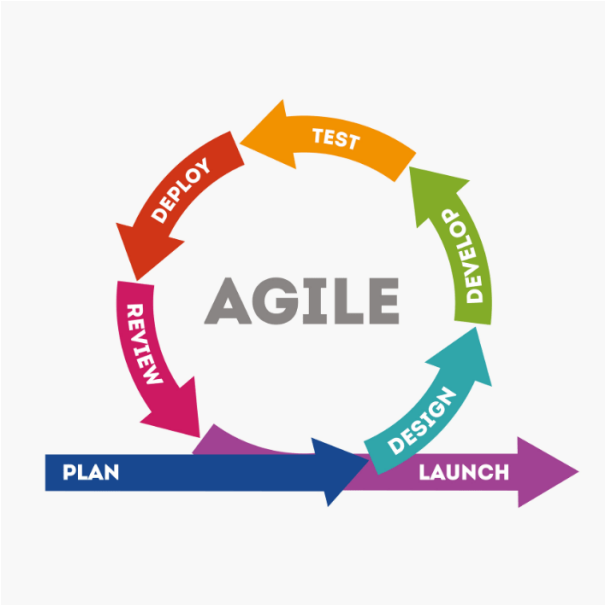
Su objetivo es verificar que los usuarios puedan realizar sus tareas con la interfaz que se propone, capturando las funcionalidades de la aplicación, sus comunicaciones e interacciones de forma adecuada.

En este caso, y a diferencia de lo que suele ser frecuente, no han sido llevados a cabo por parte del desarrollador tras la elicitación de requisitos para que el cliente valide el sistema, sino que han sido proporcionados por parte de Marina (nuestro cliente), sirviendo como guía a la hora del diseño de la aplicación.



3.4 Metodología de desarrollo

Para la elaboración de esta aplicación se ha seguido una metodología de desarrollo ágil SCRUM. Consiste en una estructura de desarrollo incremental, siendo cada iteración un sprint (entrega parcial del producto final).

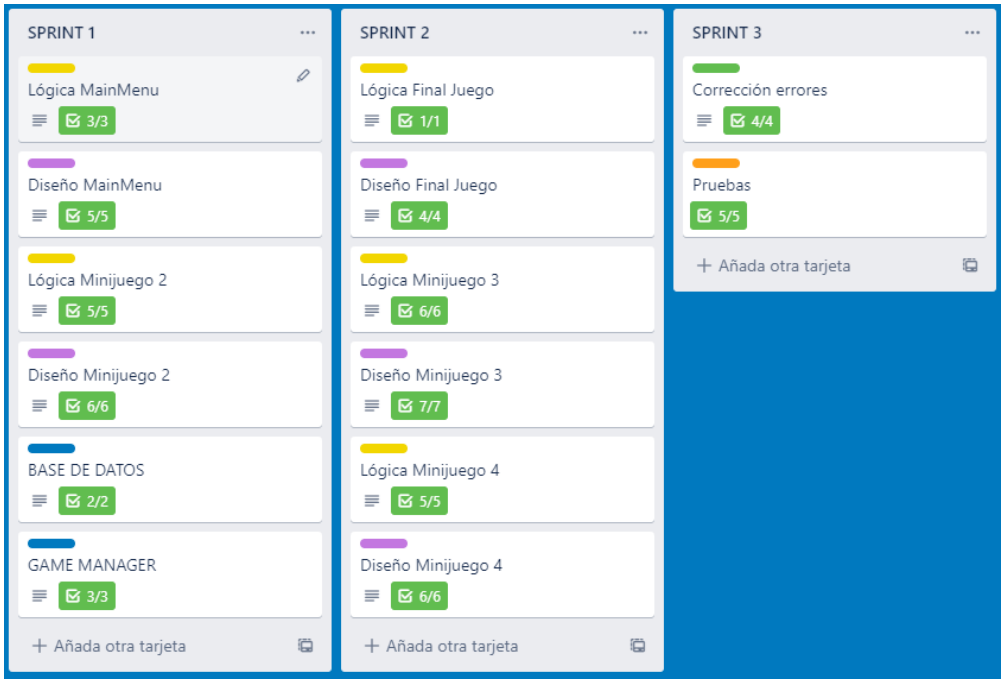


Este proyecto se ha dividido en un total de tres sprints de dos meses de duración aproximadamente, siendo la primera entrega en febrero, la segunda en abril y la última en junio.

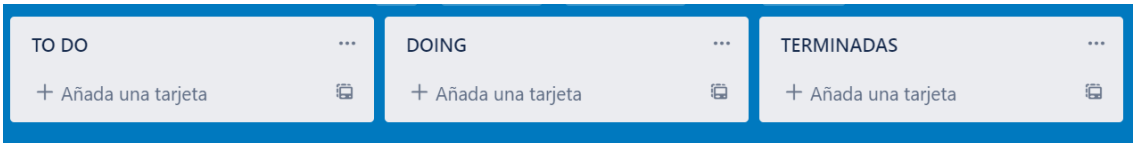
En la siguiente imagen se pueden apreciar las tareas asociadas a cada sprint. Como se puede ver, el primer sprint iba más enfocado a la preparación del control del juego, desarrollando la capa de datos (con la creación de la base de datos) y la gestión y comunicación de la capa de la lógica (con la creación de la instancia del Game Manager).

El segundo sprint fue más enfocado al desarrollo de los minijuegos como tal, con la realización de los dos últimos minijuegos.

Y finalmente el tercer sprint fue dedicado a cerrar el desarrollo con la corrección de pequeños errores de diseño determinados por Marina y realización de pruebas y la depuración de la aplicación.



Además, en la siguiente imagen se puede ver el tablero Kanban del que se ha respaldado, que sirve como herramienta para controlar y visualizar el flujo de trabajo de cada sprint, mediante la aplicación Trello.



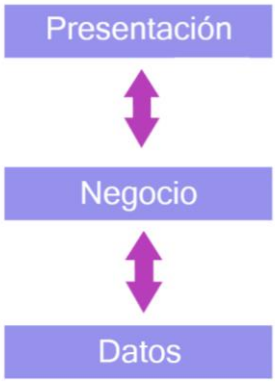
4. Diseño

En este apartado se van a detallar las decisiones tomadas para encontrar una solución apta para el proyecto, cumpliendo todos los requisitos que se han documentado anteriormente, y además se explica cómo se ha implementado esa solución.

4.1 Arquitectura

La arquitectura software define la manera de trabajar en un sistema y su estructuración. Hace referencia a la forma en que los distintos componentes del sistema software se integran entre sí. Permite una depuración de código más sencilla, reducir el riesgo, generar una independencia entre los elementos y al mismo tiempo asegurar un entendimiento común.

Esta aplicación hace uso de una arquitectura software multicapa, que consiste en la división del sistema en tres capas totalmente independientes y desacopladas unas de las otras, tal y como muestra la figura de la derecha.



Su principal ventaja es que el desarrollo se lleva cabo en varios niveles, de manera que, si se produce algún cambio, éste solo afectará al nivel al que pertenece.

4.1.1 Capa de presentación

Es la capa más cercana al usuario, aquella que éste va a visualizar y con la que va a interactuar. Corresponde a la vista de la aplicación y se comunica con la capa de negocio para indicarle las acciones del usuario.

En este proyecto ha sido desarrollada con Unity, donde cada pantalla, denominada *Escena*, está compuesta por un conjunto de objetos que constituyen el entorno y los diferentes escenarios del juego.

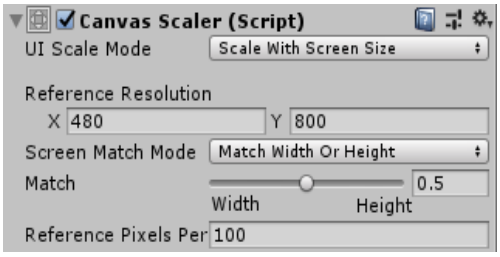
La creación de las escenas supone un proceso similar a la de ir diseñando y construyendo el juego por piezas, donde cada pieza corresponde a cada escena.

La aplicación está formada por un total de seis escenas: una pantalla de inicio como bienvenida, dos menús (de inicio y de fin) y los tres minijuegos. Todas tienen en común algunos elementos con tal de conseguir una uniformidad y coherencia entre ellas y, en consecuencia, dotar al juego en sí de una homogeneidad clara.

En primer lugar, tenemos los elementos por defecto básicos de toda escena Unity, que son la Main Camera, Event System [2] (para capturar las acciones del usuario en forma de eventos) y Audio Source (en caso de necesitar sonido o efectos).

En segundo lugar, tenemos los elementos necesarios para conseguir esa consistencia entre todas las escenas y al mismo tiempo satisfacer uno de los requisitos no funcionales más importantes, la adaptabilidad de las diferentes resoluciones posibles.

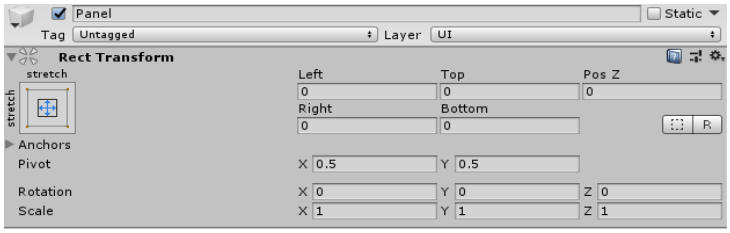
- **Canvas:** Objeto base que supone el área donde todos los elementos de la interfaz deben estar. Se le tiene que añadir el componente *Canvas Scaler* con las siguientes propiedades:

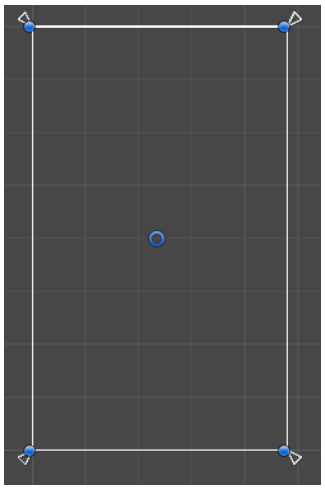


  - ***UI Scale Mode***: Determina la manera en que se escalan los elementos de la interfaz. El modo *Scale With Screen Size* hace que éstos aumenten su tamaño de forma proporcional a la pantalla.

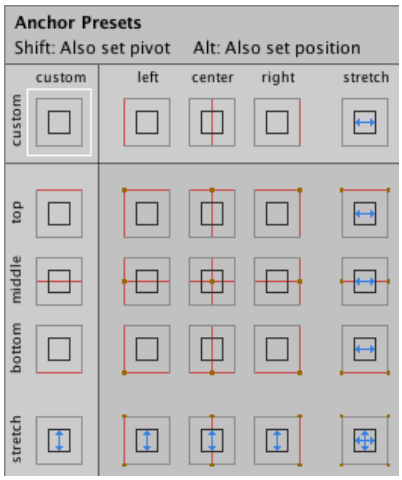
  Por defecto se elige una resolución de referencia de 480x800 píxeles para la pantalla y un porcentaje del 50% como referencia para el área del Canvas.

- **Panel**: Objeto hijo del Canvas cuya función es expandirse horizontal y verticalmente respecto a éste para ocupar todo el espacio posible. Nos sirve para que éste sea el primero que se adapte a los posibles cambios de tamaño del Canvas.





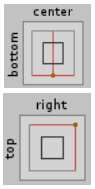
- **Anchors:** Componentes de cada elemento que permiten la localización y/o escalado del objeto en la pantalla. Estos juegan un papel muy importante porque dependido el tipo de anclaje que elijamos, los elementos se redimensionarán o no según diferentes opciones.



Por ejemplo, para los paneles del menú se ha usado un anclaje para que se extienda horizontalmente y se mantenga en el centro de la pantalla.

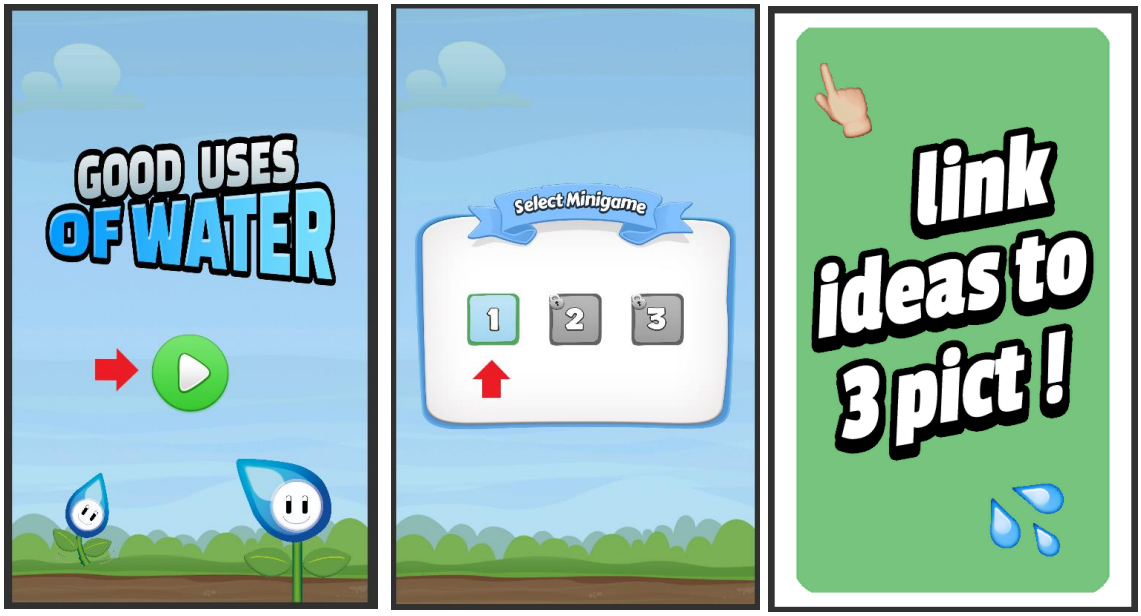
Para los grids de iconos de cada minijuego se ha utilizado un anclaje para que se extienda horizontal y verticalmente.

Y por el contrario, para el cronómetro se ha utilizado un anclaje para que se mantenga en la esquina derecha o centrada abajo, sin redimensionarse.



1. Ejemplo de uso

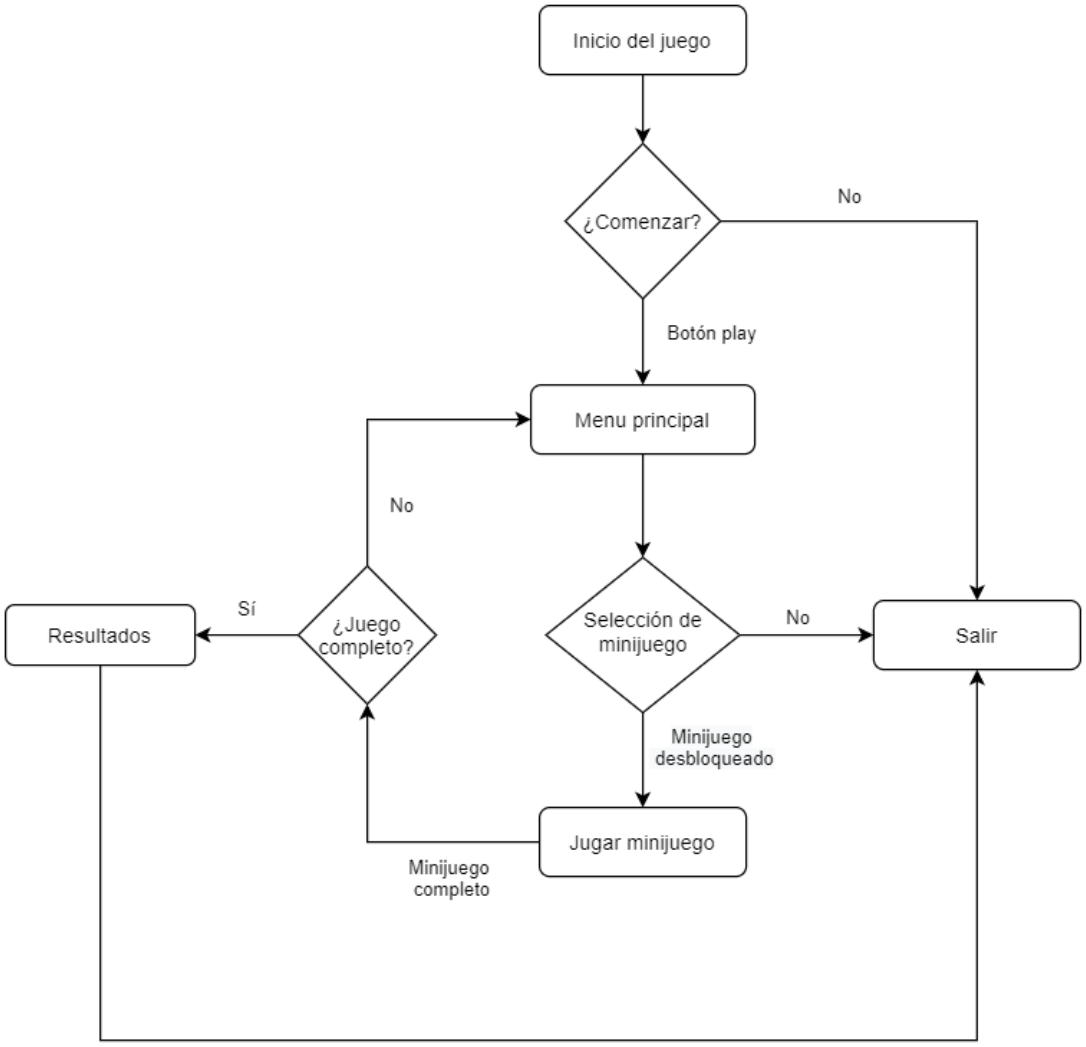
Esta sería la secuencia de pasos en el juego para iniciar el primer minijuego:



Tras darle al botón Play de la escena inicial nos lleva al menú principal con la selección de niveles. En este menú solo nos da la opción de seleccionar el primer minijuego, ya que los otros están bloqueados todavía.

Al darle al botón del primer minijuego nos lleva a la escena de éste, dando comienzo entonces el minijuego.

El transcurso del videojuego completo está recogido en el siguiente diagrama de flujo.



4.1.2 Capa de negocio

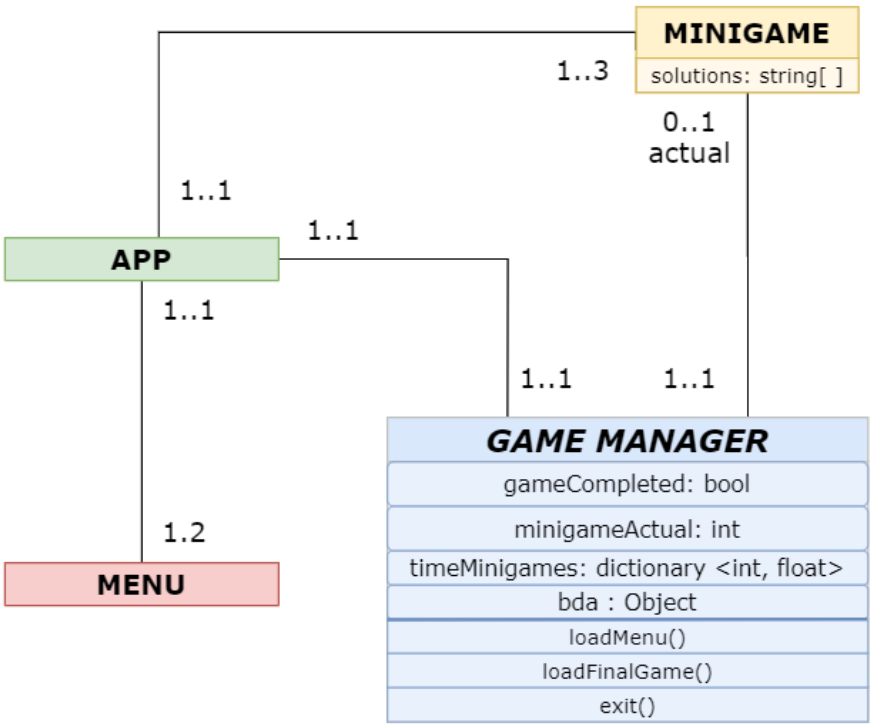
Es la capa intermedia donde se encuentran todas las funcionalidades que se ejecutan. Es la encargada de recibir las acciones del usuario que envía la capa de presentación, procesar la información comunicándose con la capa de datos (si es necesario) y enviar posteriormente la respuesta de nuevo a la capa de presentación.

En esta aplicación encontramos un total de nueve clases: una por cada escena del juego, dos clases que permiten la interacción del usuario en cuanto a las acciones de *Drag & Drop* (arrastrar y soltar), una clase para todo lo referido a la base de datos y finalmente la clase para el *Game Manager*, el núcleo administrador para todas ellas y del juego en sí.

Uno de los pilares más importantes del desarrollo de videojuegos es de la existencia del mencionado Game Manager [4], un objeto cuyo objetivo principal es la de mantener todos los elementos comunes al estado del juego (como puntuaciones, nivel actual, personaje seleccionado…) y funcionalidades (como guardar o cargar una partida, cambios de escena…) alojados en el mismo lugar y accesibles en cualquier punto del programa.

De esta manera, la información de si el juego está completo, el minijuego actual en el que se encuentra el usuario, el tiempo empleado en finalizar cada minijuego, etcétera puede ser consultada y actualizada en cualquier momento y permite tener un control de los cambios producidos entre las diferentes escenas.

La siguiente figura representa el diagrama de clases obtenido para esta aplicación.



4.1.3 Capa de datos

Es la capa donde residen los datos y la única encargada de acceder a éstos. Puede estar compuesta por uno o varios gestores de datos que efectúan todo el almacenamiento de los datos pertinentes. Reciben las solicitudes de recuperación o actualización de información desde la capa de negocio.

En esta aplicación la capa de datos es muy simple, pues se trata de un objeto compuesto por tres atributos, uno por cada minijuego, que contienen las soluciones de éstos y métodos consultores para poder acceder a ellos desde fuera en el momento de comprobar si la solución propuesta es correcta.

En un principio iba a ser una solución más compleja donde las soluciones se encontraban en un archivo JSON al cual se accedía desde esta misma clase. En este caso los atributos eran diferentes y únicamente se trataba de una lista de objetos creados para esta cuestión. Este nuevo objeto contenía un identificador del minijuego y sus soluciones asociadas. Sin embargo, la incompatibilidad del formato JSON con los dispositivos móviles imposibilitó esta vía.

Por último, después de lo mencionado anteriormente, cabe destacar también que esta capa no está destinada a la persistencia de datos como tal, pues el usuario no almacena ningún tipo de información fuera del transcurso del juego en ese preciso instante. Es decir, no se guarda en ningún momento información personal del usuario ni tampoco el último estado del juego.

Esta capa únicamente sirve para separar la cuestión de las soluciones de la lógica del programa y permitir la consulta de éstas.

4.2 Patrones de diseño

Los patrones de diseño se pueden definir como una forma reutilizable estandarizada de resolver problemas frecuentes en el desarrollo y comunes entre diferentes componentes. Son útiles porque ayudan a ahorrar tiempo, establecer un lenguaje colectivo, establecer validez al código…

Se pueden clasificar en tres tipos según la finalidad asociada: creacionales (solucionan las tareas de creación o instanciación de objetos), estructurales (definen la composición de los objetos) y de comportamiento (establecen la comunicación entre los objetos).

En esta aplicación sólo se ha hecho uso de un patrón de diseño de tipo creacional, el cual se detalla en profundidad a continuación.

4.2.1 Singleton

Este patrón se utiliza cuando, por algún motivo, se necesita que sólo exista una instancia de una determinada clase y que se tenga un acceso global a ésta en toda la aplicación.

Esta definición corresponde exactamente con el concepto del *Game Manager* en nuestra aplicación, donde uno de sus requisitos primordiales es que solo debe de haber una instancia de éste durante todo el transcurso del juego. Será esta instancia la que irá pasándose entre las diferentes escenas con el fin de evitar problemas de inconsistencia [4].

Para su correcto funcionamiento requiere un constructor de acceso privado y métodos de acceso público de solo lectura (consultores) y de escritura para la actualización de sus variables (modificadores).

A continuación, se muestra su implementación en el proyecto:

1. Atributos y métodos públicos: Se declara el objeto de la clase que corresponderá a la instancia en cuestión, además de los atributos necesarios (detallados en el diagrama de clases del apartado de la capa de negocio en este mismo apartado). Los métodos públicos *ActualMinigame* y *TimeMinigame* hacen uso de las variables privadas correspondientes.

```
#region Game
  public static GameManager Instance {get; private set; }
  public bool GameCompleted;
  public BDA bda;
  private int _actualMinigame;
  public int ActualMinigame {
      get { return _actualMinigame; }
      set {_actualMinigame = value; }
  }
  private Dictionary<int,float> _timeMinigames;
  public Dictionary<int,float> TimeMinigames {
      get { return _timeMinigames; }
      set {_timeMinigames= value; }
  }
#endregion
```

2. Constructor: Se hace uso de la instancia estática de la clase donde, se crea y se impide su destrucción si se accede a ella por primera vez, o se destruyen las posibles copias existentes si se accede el resto de las veces.

Hay que destacar que el método *Awake* es el primero que se ejecuta dentro del ciclo de vida del script y solo una vez. Se suele utilizar para la inicialización de las variables y objetos del juego antes de empezar.

```
void Awake() {
  if (Instance == null) {
        Instance = this;
        DontDestroyOnLoad(gameObject);
   } else {
        Destroy(gameObject);
     }
}
```

5. Tecnologías y herramientas

En este apartado se van a describir todas las tecnologías y herramientas de las que se ha hecho uso a lo largo del proyecto.

5.1 Plataformas de desarrollo

Unity es el entorno de desarrollo empleado en este proyecto. Es un motor de videojuegos que permite la creación, el diseño y el funcionamiento de videojuegos 2D o 3D para diversas plataformas mediante un editor visual y programación vía scripting [12]. Fue fundado por Unity Technologies en el año 2005.

Entre sus herramientas más destacadas se encuentran el soporte a la inteligencia artificial, la navegación NavMesh, la realidad virtual… Además, cuenta con la posibilidad de poder manejar de forma más sencilla el código usando el sistema de control de versiones.

Visual Studio Code es el otro entorno de desarrollo empleado en este proyecto, dedicado a la programación de la aplicación [14]. Se trata de un editor de código fuente que incluye soporte de depuración, control integrado Git, resaltado de sintaxis… Fue fundado por Microsoft en el año 2015

Una de sus principales potencias es la posibilidad de instalación de múltiples plugins, bien para la realización del trabajo en sí, o bien para facilitar éste.

5.2 Lenguajes de programación

C# es el lenguaje de programación empleado en el proyecto [13]. Se trata de un lenguaje multiparadigma diseñado para la infraestructura de lenguaje común. Fue fundado por Microsoft en el año 2000, como parte de su popular framework .NET, y ha sido estandarizado por ECMA e ISO.

Su sintaxis deriva de C/C++, presenta seguridad de tipos y utiliza modelado de objetos (similar al usado en el lenguaje Java) y de componentes, tales como eventos, propiedades…

5.3 Herramientas

Draw.io es la herramienta empleada en este proyecto para la realización de los diagramas adjuntados en este documento. Se trata de una aplicación web gratis que permite crear y editar diagramas de todo tipo, desde cero o a partir de modelos otorgados por la propia herramienta. Fue fundado por JGraph en el año 2000.

Adobe Photoshop es la herramienta empleada en este proyecto para la edición de los componentes de la interfaz de la aplicación. Se trata de un editor de imágenes, gráficos, tanto rasterizados como vectoriales, en 2D o 3D. Fue fundado por Adobe Systems Incorporated en el año 1990**.**

Trello es la herramienta empleada en este proyecto para la gestión y planificación de las tareas de la aplicación. Se trata de un software de administración de proyectos online que permite aclarar y mejorar la rutina de trabajo y la organización de éste basándose en el método Kanban. Fue fundado por Atlassian en el año 2011.

5.4 Recursos y librerías

Una de las principales potencias y ventajas de Unity es la existencia de la *Asset Store* [11], una biblioteca accesible a los usuarios donde se pueden encontrar todo tipo de recursos, gratis y de pago, para utilizarlos en sus proyectos. Presenta varios tipos de *assets* que van desde texturas, animaciones, modelos, imágenes, sonidos… incluso tutoriales y extensiones para el propio programa.

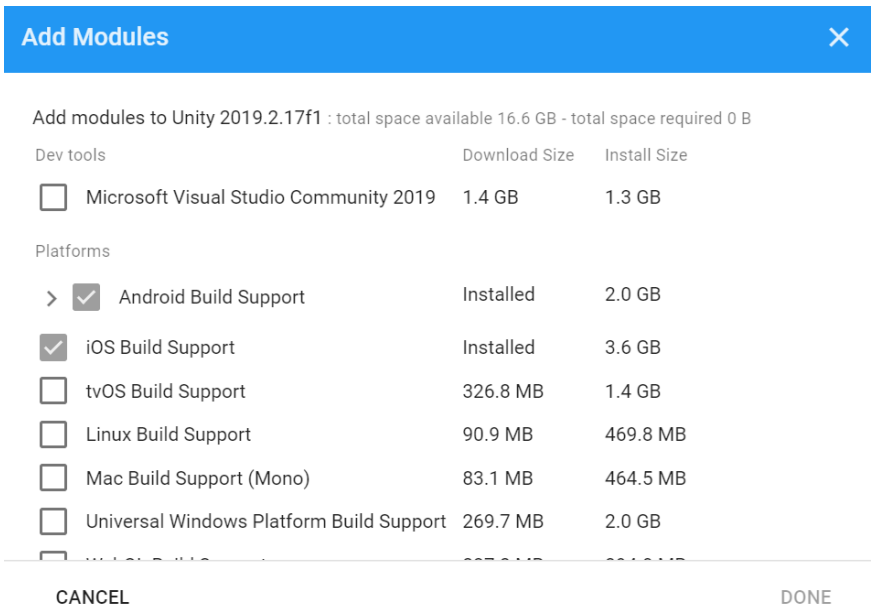
En este proyecto se han utilizados recursos de tipo multimedia para el desarrollo de la interfaz de cada escena de la aplicación. En concreto de *SuperBaseball* para la creación de los títulos necesarios (pantalla principal y presentación de los minijuegos) y de *Puzzle stage & settings GUI Pack* para el uso de determinados iconos.

Además, para la realización de las pruebas de la aplicación se ha empleado el *framework* ***NUnit*** de Microsoft NET, que únicamente necesita su importación en la clase a la hora de su uso.

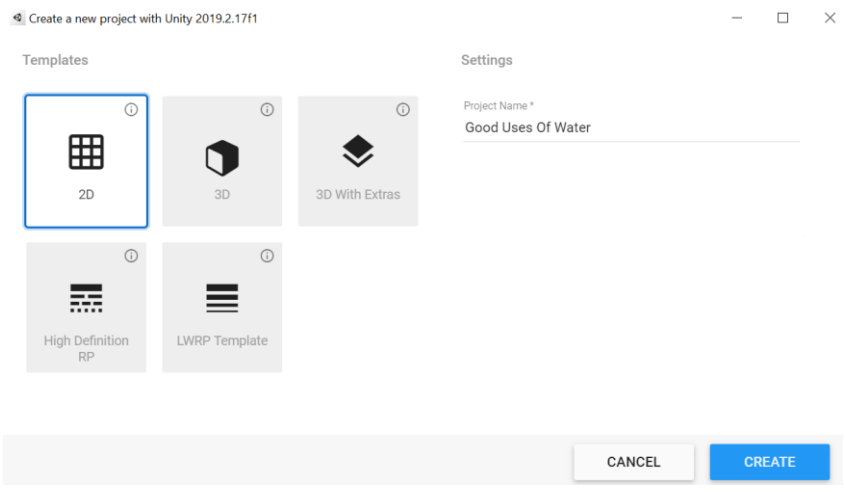
5.5 Preparación del entorno de desarrollo

5.5.1 Instalación de Unity

El primer paso es la instalación del programa Unity, que se puede descargar desde la propia página oficial de la compañía. Se ha hecho uso de la versión Unity 2019.2.17f1, con la instalación de los módulos que se ven a continuación para poder realizar la exportación de la aplicación en esas plataformas de destino.



Para la creación del proyecto se ha elegido el template 2D, puesto que para esta aplicación no es necesaria la presencia tridimensional.

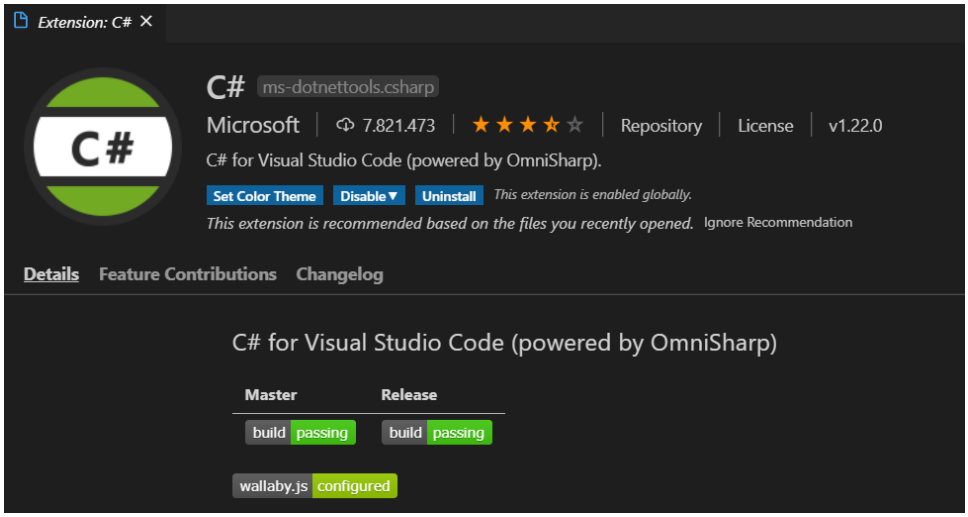


5.5.2 Instalación de Microsoft NET Core SDK

Para poder trabajar con el lenguaje C# es necesaria la previa instalación de la última versión del framework .NET Core Software Development Kit de Microsoft. La descarga se puede encontrar en la página oficial de la compañía, la última versión estable que se encontraba en ese momento, y por lo tanto la utilizada en el proyecto, ha sido la 3.1.100.

5.5.3 Instalación de C# (Visual Studio Code)

Una vez instalado el framework de Microsoft se procede a la instalación del lenguaje de programación C# [9], necesario para llevar a cabo la programación vía script de Unity. En este caso, dado que el entorno de desarrollo utilizado es Visual Studio Code (previamente instalado también), ésta se efectúa desde el propio programa (en concreto en la ventana de extensiones). De forma automática se descarga e instala la última versión estable, contando con un soporte de actualización inmediato.



6. Implementación

En este apartado se van a describir los aspectos más destacados en cuanto a la implementación de la aplicación, en concreto se explicará más detalladamente puntos clave del desarrollo y algunos cambios producidos durante éste.

6.1 Estructura del proyecto

Al crear un nuevo proyecto Unity establece una estructura por defecto de diferentes paquetes destinados a recursos, la cual se ha mantenido. No obstante, se han añadido nuevos paquetes para garantizar una organización y tener buenas prácticas durante el desarrollo.

- Animations: Contiene los elementos dedicados a las animaciones de la aplicación, en relación con las transiciones que se producen entre los cambios de escenas.

- Prefabs: Contiene todos los *prefabs* existentes en la aplicación. Son objetos reutilizables creados con una serie de características definidas y que son instanciados varias veces durante el juego.

- Resources: Contiene todos los recursos multimedia de los que se hace uso durante el juego. Este paquete a su vez se encuentra dividido entre los diferentes tipos de recursos que podemos encontrarnos y su finalidad (imágenes, sonidos, fuentes de texto…).

- Scenes: Contiene todas las escenas creadas que aparecen en el juego.

- Scripts: Contiene todos los scripts generados para implementar las funcionalidades que necesita la aplicación. Este paquete a su vez se clasifica según al sector del juego al que pertenece.

- Tests: Contiene todas las pruebas que se han realizado para garantizar el correcto funcionamiento de la aplicación.

6.2 Implementación por pasos

A continuación, se van a detallar los aspectos más complejos y destacables del desarrollo completo de la aplicación. Por comodidad y facilitar la compresión se van a tratar por orden de aparición en el juego.

6.2.1 Main Menu

El menú principal es el primer elemento que aparece en la aplicación, donde el usuario puede ver el minijuego actual en el que se encuentra y comenzar a jugar.

En el script asociado se inicializa por primera y única vez el Game Manager en el juego (marcando el principio del juego), además de los componentes de la escena.

La interfaz del menú muestra los minijuegos como bloqueados, desbloqueados o completos en función del progreso del juego que presente el usuario en ese momento.

En la imagen anterior, que servirá como ejemplo de explicación, el usuario se encuentra en el minijuego 2, por lo tanto, el minijuego 1 se muestra como completo y el minijuego 3 como bloqueado.

Esta información se obtiene a través del número del minijuego actual almacenado en el Game Manager, tal y como se aprecia a continuación.

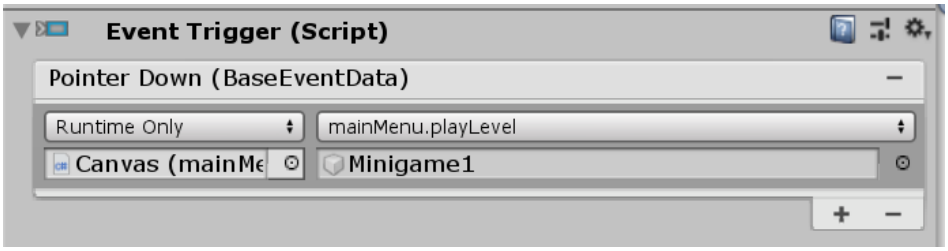
```
void inicializateUI() {
    int actualMinigame = gameManager.ActualMinigame;
    if(gameManager.GameCompleted) gameCompleted();
    else {
        if (1<actualMinigame) {
            changePhoto(minigame1, true);
            star1.SetActive(true);
        }
        if (2<= actualMinigame ) {
            changeInteractable(minigame2);
            if (2<actualMinigame) {
                changePhoto(minigame2, true);
                star2.SetActive(true);
            }
            else changePhoto(minigame2, false);
        }
        if (3<= actualMinigame) {
            changeInteractable(minigame3);
            if (3<actualMinigame) {
                changePhoto(minigame3, true);
                star3.SetActive(true);
            }
            else changePhoto(minigame3, false);
        }
    }
}
```

1. La instrucción ***star.setActive(true)*** hace que aparezca la imagen de la estrella en aquellos minijuegos que estén completos, activando el Game Object de la escena.
2. El método ***changePhoto*** es el que permite el cambio de imagen en el minijuego en función del estado en el que se encuentre éste. Con el booleano se indica si el minijuego está completo o no.
3. El método ***changeInteractable*** es el que permite que el usuario pueda interaccionar con el botón del minijuego en cuestión, activando el componente *Toggle* del Game Object asociado, que en un principio está desactivado para impedir el acceso a los minijuegos bloqueados.

```
public void changeInteractable(GameObject minigame) {
    minigame.GetComponent<Toggle>().interactable=true;
}
```

Siguiendo con el ejemplo, la figura de arriba es el caso del minijuego 3 (bloqueado) y la de abajo el caso del minijuego 1 (accesible desde el comienzo) y del minijuego 2 (cambia en ejecución con el método anterior).

De hecho, en el momento en el que se produce el evento de pulsar el botón de un minijuego para jugarlo, se ejecuta la siguiente instrucción, en la cual solo si este componente está activado, se carga la escena del minijuego seleccionado.




```
public void playLevel(GameObject pulsed) {
    if(pulsed.GetComponent<Toggle>().interactable==true)
        SceneManager.LoadScene(pulsed.name);
}
```

6.2.2 Minigame 1

El primer minijuego consiste en seleccionar los tres pictogramas que se corresponden con la actividad que se pide. Está compuesto por cinco niveles que hacen relación a acciones sobre el agua, como lavarse las manos, beber, reciclar agua…

En el script asociado, tras instanciar los componentes de la escena, lo primero que se realiza es la preparación de ésta para el juego. Con tal de incrementar la dificultad y dotar al minijuego de innovación, al principio de cada nivel se ejecuta un algoritmo de mezcla entre los iconos, intercambiando las imágenes aleatoriamente y evitando así que éstos siempre estén colocados en el mismo sitio.

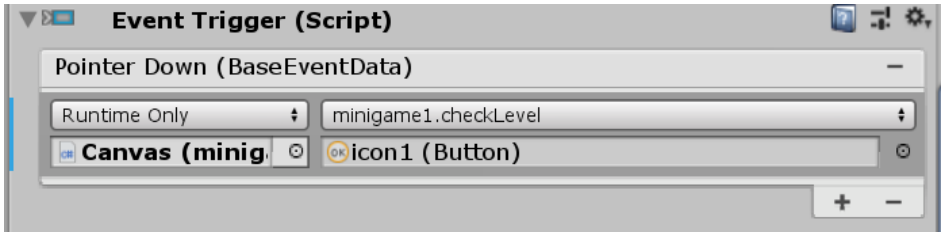
En el momento que se produce el evento de pulsar un botón se comprueba que ese icono no haya sido pulsado antes para añadirlo al array de soluciones del usuario y mostrarlo como pulsado (*interactable=false*).

El cambio visual de los iconos al realizar esta acción se muestra en las siguientes dos imágenes.

Por otra parte, cuando el usuario ha seleccionado los tres iconos que se piden se comprueba si la solución propuesta es correcta (comparándolo con las soluciones). Si lo es, se para el tiempo para mostrar la solución, se reproduce el sonido de acierto y se pasa al siguiente nivel.

El último paso es reiniciar la solución del usuario, mostrando los botones sin pulsar y el contador a cero con el fin de que el juego siga su curso como al principio.

Se puede ver en el siguiente fragmento de código:




```
public void checkLevel(Button pulsed) {

    bool correctSolution = true;

    if (!solutionUser.Exists(button => button.image.sprite.name == pulsed.image.sprite.name)) {
        solutionUser.Add(pulsed);
        pulsed.interactable = false;
    }

    if (solutionUser.Count == 3) {
        for (int i = 0; i < solutionUser.Count && correctSolution; i++)
            if (!solutionUser[i].image.sprite.name.Contains(solutions[level])) correctSolution = false;

        if (correctSolution) {
            controlTime(true);
            viewCanvas(canvasSolution);
            playSound("acierto");
            level++;
        }

        for (int i = 0; i < solutionUser.Count; i++) solutionUser[i].interactable = true;
        solutionUser.Clear();
    }
}
```

6.2.3 Minigame2

El segundo minijuego consiste en seleccionar cinco iconos (emojis) que hagan referencia a buenos usos del agua en la vida cotidiana (cerrar el grifo, ducharse, la lluvia...). Dichos iconos se tienen que arrastrar desde su posición hacia un círculo y éste desechará los incorrectos y se iluminará cuando contenga los cinco correctos.

Para conseguir el movimiento de arrastre de los iconos hacia el círculo se ha hecho uso de otro Game Object que va a representar el icono que se lleva en la mano el usuario en ese momento (llamado *hand*) y de una serie de componentes y scripts (independientes al de la escena), los cuales se detallan a continuación.

Los componentes que se precisan en la escena para garantizar el correcto funcionamiento del juego son:

- **Box Collider 2D:** Collider en forma de rectángulo que presenta una posición y dimensiones de anchura y altura y el responsable de detectar la colisión física de un elemento.
- **Rigidbody 2d:** Coloca un objeto bajo el control del motor de física, permitiendo su movimiento, su rotación, tener gravedad, masa… Además, es el elemento de conexión entre diferentes colliders.

Los scripts que se han desarrollado se han centrado en capturar un conjunto de eventos de los componentes colliders de la escena durante el transcurso del minijuego.

En primer lugar, está el script *DragDrop* que se encarga de controlar el movimiento de arrastre de los iconos desde el punto de vista del origen, por lo tanto, está asignado a éstos y al objeto *hand*. Maneja los siguientes eventos:

- **OnBeginDrag:** Este evento se lanza en el momento que el usuario comienza el arrastre del icono. Se realiza un intercambio entre el icono seleccionado y el objeto de la mano, haciendo visible este último (que en un principio tiene la opacidad a cero).

```
hand.GetComponent<Image>().sprite = item.GetComponent<Image>().sprite;
hand.GetComponent<Image>().color = new Color (1f,1f,1f,1f);
```

- **OnDrag:** Este evento se lanza mientras el usuario sigue manteniendo el movimiento de la mano, por lo que se va actualizando la posición de ésta instantáneamente con la siguiente instrucción:

```
hand.transform.position = data.position;
```

- **OnEndDrag:** Este evento se lanza en el momento que el usuario finaliza el movimiento de arrastre del icono. Si el sitio donde se ha soltado el elemento no es el círculo, se desecha la acción y la mano vuelve a tener la opacidad a cero. Esta información se obtiene a través de una consulta a la variable *valid* del otro script (*Slot)*, que es la que proporciona la información de si el destino es el correcto o no.

```
if (!Slot.valid) hand.GetComponent<Image>().color = new Color(1f,1f,1f,0f);
```

En segundo lugar, está el script *Slot* que se encarga de controlar el movimiento de arrastre de los iconos desde el punto de vista del destino, por lo tanto, está asignado únicamente al círculo. Maneja el siguiente evento:

- **OnDrop**: Este evento se lanza en el momento que el usuario suelta el icono que tenía en la mano. Es el único punto donde la variable valid se asigna a true, indicando que el destino es correcto. Después se retoma la ejecución del script del minijuego para que siga su curso y por último se quita la imagen que tenía asociada la mano en ese momento.

```
valid=true;
minigame2.checkIcon(hand.GetComponent<Image>().sprite);
hand.GetComponent<Image>().sprite = null;
hand.GetComponent<Image>().color = new Color(1f,1f,1f,0f);
```

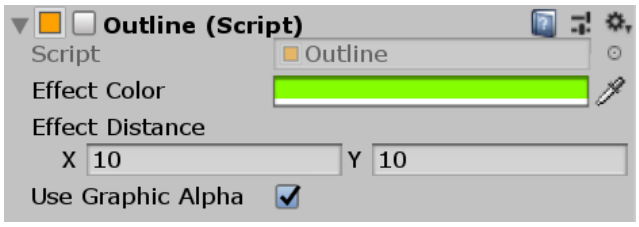
Por otra parte, en el script principal asociado a la escena se tiene que añadir el icono que se tiene en la mano a las soluciones del usuario, siempre y cuando ese icono no esté ya incluido. Por defecto, las soluciones del usuario tienen asignada una imagen secundaria invisible llamada *circle* que sirve para detectar el número de soluciones que lleva el usuario hasta el momento, por lo que solo se tiene que intercambiar esa imagen con la nueva que quiere añadir y hacerla visible dentro del círculo.

Cuando las soluciones del usuario llegan a las cinco que se piden, se pasa a comprobar si la solución propuesta es correcta.

```
public static void checkIcon(Sprite icon) {
    bool cont = true;
    if (!Array.Exists(solutionUser, solution => solution.sprite.name == icon.name)) {
        for (int i=1;i<solutionUser.Length && cont;i++) {
            if(solutionUser[i].sprite.name == "circle") {
                solutionUser[i].sprite = icon;
                solutionUser[i].color = new Color(1f,1f,1f,1f);
                cont = false;
                contSolution++;
            }
            if(solutionUser[i].sprite.name == icon.name) cont= false;

            if(contSolution==5) checkSolution();
        }
    }
}
```

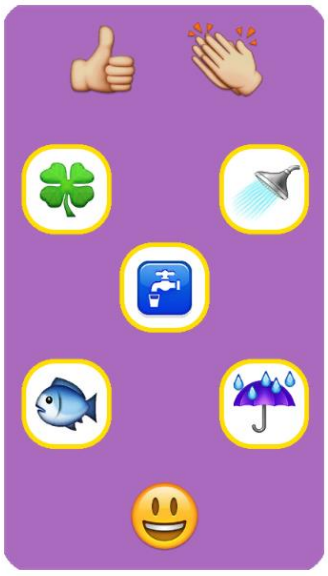
Para ello se comprueba con las soluciones del juego si en la solución propuesta existen iconos que no sean correctos, eliminándolos del círculo (volver a asignarles la imagen secundaria y hacerlos invisibles), y manteniendo los correctos para continuar el transcurso del juego.

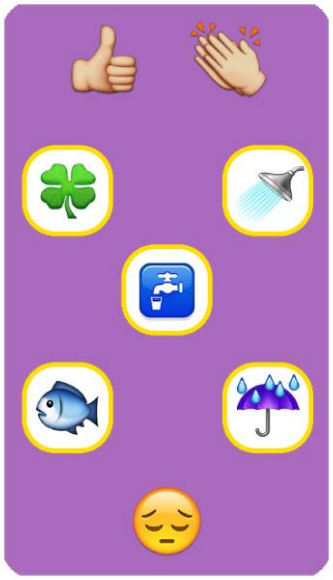
En el caso de que la solución sea correcta se activa el componente *Outline* (borde sombreado de color verde) asociado al círculo para indicar que la solución es correcta.



También se lleva un conteo del número de intentos del usuario para completar el minijuego, con tal de otorgarle una calificación de muy bien, bien o regular en forma de emojis.

```
static void checkSolution() {
    bool test =true;
    intent++;

    for(int i=1; i < solutionUser.Length; i++) {
        if (!Array.Exists(solutions, solution => solution == solutionUser[i].sprite.name)) {
            correctSolution = false;
            solutionUser[i].sprite= Resources.Load<Sprite>("Images/Minigame2/circle");
            solutionUser[i].color = new Color(1f,1f,1f,0f);
            test = false;
            contSolution--;
        }
        if(i==solutionUser.Length-1 && test) correctSolution = true;
    }

    if(correctSolution) solutionUser[0].gameObject.GetComponent<Outline>().enabled = true;
}
```





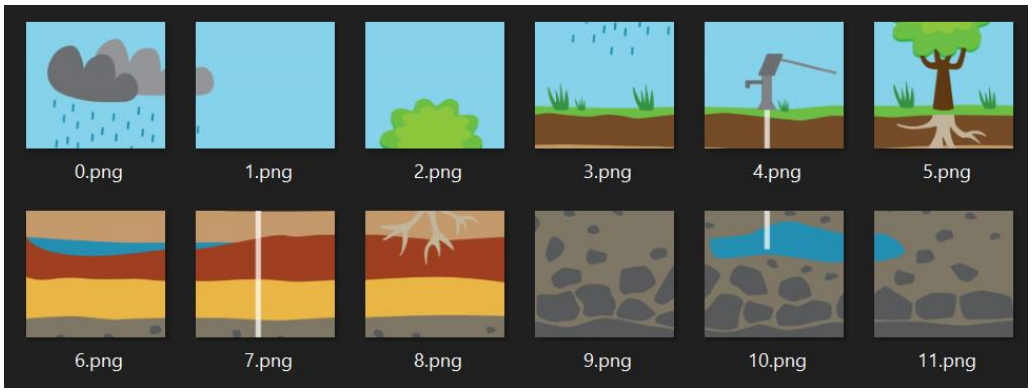
Otro aspecto importante es que los iconos que se muestran van cambiando cada segundo, de manera que el usuario los ve pasar uno a uno. Para conseguir esto se inicia la ejecución en bucle de una corrutina que solo se detiene cuando la solución es correcta, con las siguientes instrucciones.

```
InvokeRepeating("changePhoto", 0f, 1.0f);        CancelInvoke("changePhoto");
```

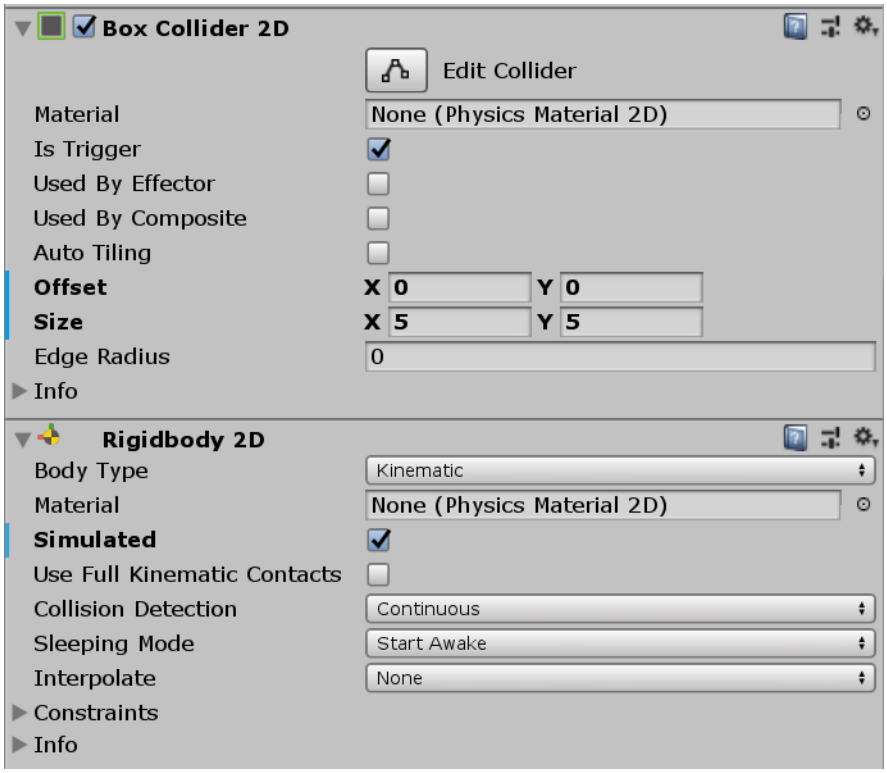
6.2.4 Minigame3

El tercer y último minijuego consiste en un puzzle de doce piezas que el usuario tiene que completar. Hace referencia a una imagen sobre el ciclo del agua y de dónde viene ésta.

Para conseguir las piezas se ha hecho uso de una herramienta online y gratuita de fragmentación de una imagen en doce.



Estos doce elementos, además del Game Object que corresponde con la pieza actual que el usuario intenta colocar, precisan de dos componentes en la escena para garantizar el funcionamiento y que son el principal eje del juego: el **Box Collider 2D** y el **Rigidbody 2D** (mencionados y detallados en el minijuego anterior). En este caso es necesario que todos los collider tengan activada la característica de *Is Trigger* para que se permita atravesarlo en el momento de la colisión.



Además, también hacen uso del script *DragDrop* para detectar el movimiento de las piezas. En este minijuego, se capturan los siguientes eventos:

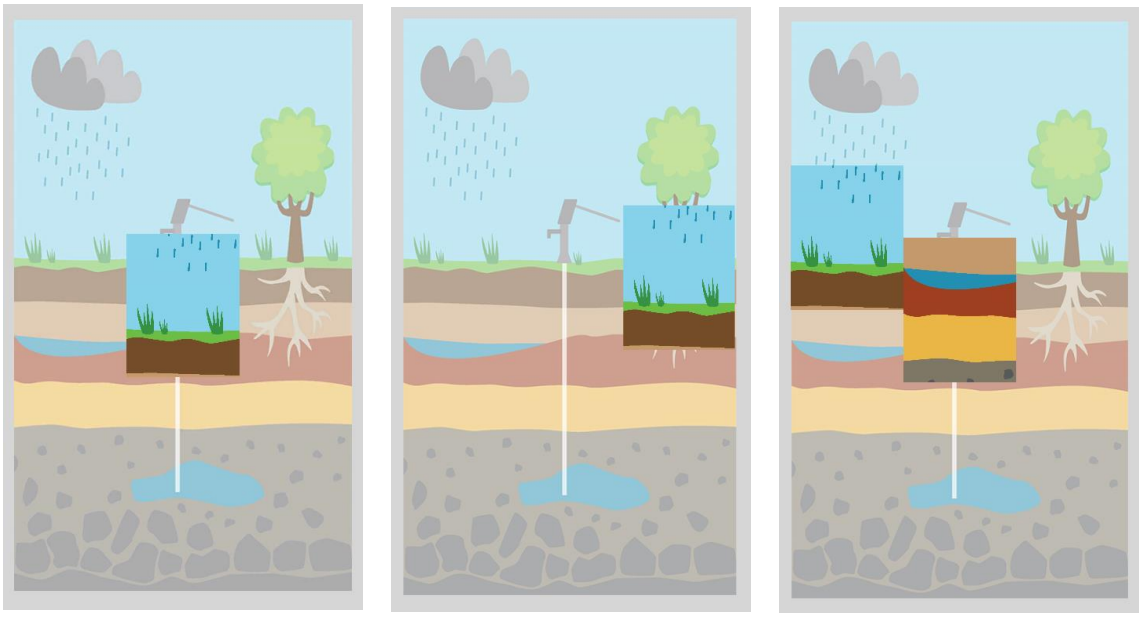
- **OnDrag:** Este evento se lanza mientras el usuario sigue manteniendo el movimiento de la pieza, por lo que se va actualizando la posición de ésta y la de su collider asociado instantáneamente con las siguientes instrucciones.

```
item.transform.position = data.position;
colliderItem.transform.position = data.position;
```

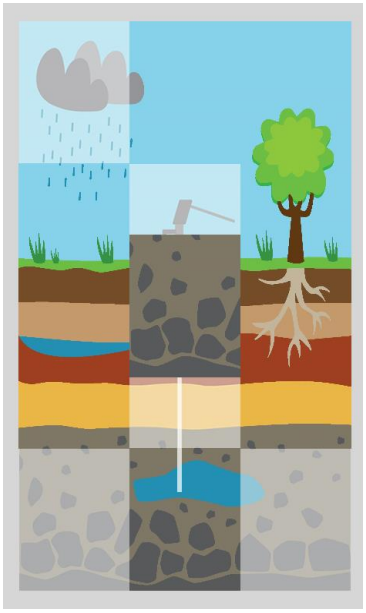
- **OnTriggerStay2D:** Este evento se lanza en el momento que el collider de la pieza actual entra en contacto con el collider de alguna de las doce piezas del puzzle. En ese caso se inicia la ejecución del script del minijuego para que el juego siga su curso.

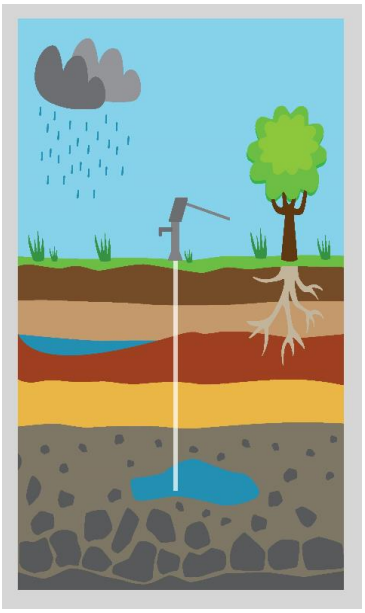
```
minigame3.checkPiece(collision.name);
```

Por otra parte, en el script principal asociado a la escena se tiene que comprobar que el lugar donde se ha colocado la pieza actual sea el que corresponde. En ese caso la pieza del puzzle se hace visible y se escoge aleatoriamente la siguiente pieza actual, que aparecerá en el centro de la pantalla.



En este caso, que servirá como ejemplo explicativo, se obtiene al inicio la pieza número 3. En la segunda y tercera imagen se puede ver como la nueva pieza no sale hasta que el lugar donde se coloca la pieza actual es el correcto.





Siguiendo con el ejemplo expuesto, en las anteriores imágenes se puede ver la evolución del minijuego a raíz de que el usuario va colocando una a una las piezas hasta que llega a completarlo (última imagen).

Por otra parte, en las siguientes imágenes se muestra este proceso en orden de ejecución:

```
public static void checkPiece(string pieceCollision) {
    if (pieceCollision == namePiece) nextPiece();
}
```

```
static void nextPiece() {
    pieces[pieceActual].color = new Color(1f,1f,1f,1f);
    changes=true;
}
```

```
void newPiece() {
    piece.color = new Color(1f,1f,1f,0f);

    if (randomList.Count>0){

        Thread.Sleep(500);
        pieceActual = randomList[0];
        randomList.RemoveAt(0);
        piece.transform.position = centerPosition;
        piece.sprite = Resources.Load<Sprite>("Images/Minigame3/pieces/"+pieceActual);
        piece.color = new Color(1f,1f,1f,1f);
        namePiece=pieces[pieceActual].name;
        pieces[pieceActual].gameObject.GetComponent<BoxCollider2D>().enabled=true;

        changes=false;
    }

    else {
        piece.gameObject.SetActive(false);
        changes = false;
        minigameCompleted();
    }
}
```

El uso de la instrucción ***Thread.Sleep(500)*** de la última imagen de código sirve para dotar al juego de más realismo, pues detiene el juego durante medio segundo en el instante que se posiciona la pieza actual en su sitio y se retoma con ésta ya colocada, apareciendo una nueva pieza a ubicar. El usuario ve una ligera pausa en el juego que no llama la atención al ser muy breve y es capaz de detectar enseguida que ha puesto la pieza correctamente.

6.2.5 Final Menu

El menú del final es el último elemento que aparece en la aplicación, donde el usuario puede ver sus resultados en el juego. Consiste en una especie de clasificación de los minijuegos según el tiempo empleado en completarlos, ordenados de menor a mayor.

El tiempo de cada minijuego está almacenado en una variable de tipo diccionario cuya clave es el número del minijuego (tipo entero) y su valor es el tiempo empleado (tipo número real) contenida en el Game Manager, por lo tanto, se realiza una ordenación ascendente por valor de la siguiente manera.

```
times = gameManager.TimeMinigames;
var t = from pair in times orderby pair.Value ascending select pair;
```

Después simplemente tiene que recorrerse esa variable para preparar la escena con esta información y cargar las imágenes de los minijuegos en el orden correcto.

```
int i=0;
foreach (KeyValuePair<int, float> pair in t) {
    Classification[i].GetComponentInChildren<Text>().text = convert(pair.Value);
    Classification[i].sprite = Resources.Load<Sprite>("UI/MainMenu/Minigame"+ pair.Key);
    i++;
}
```

En esta imagen se puede ver un ejemplo de lo que se ha explicado anteriormente, en la que se muestra el resultado final del usuario y donde se ha realizado correctamente la ordenación y la asignación de las imágenes correspondientes a los minijuegos.

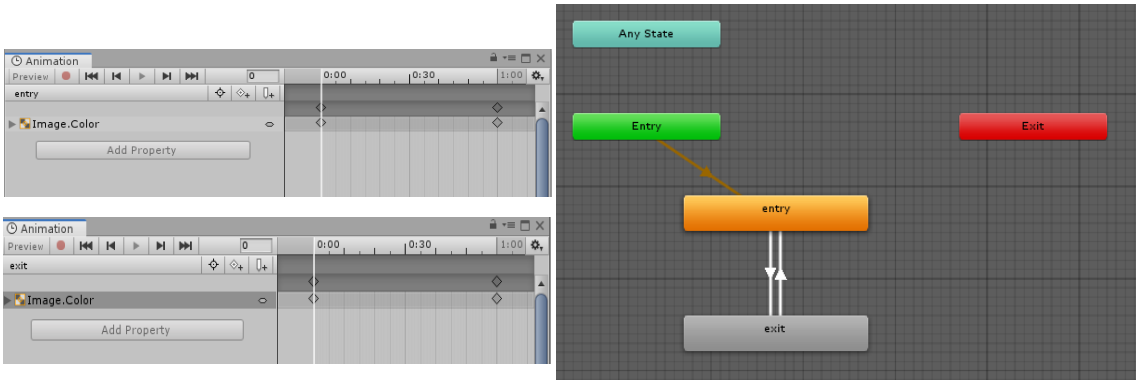
6.2.6 General

Durante el juego hay tres patrones de ejecución que tienen lugar constantemente y que también hay que comentar.

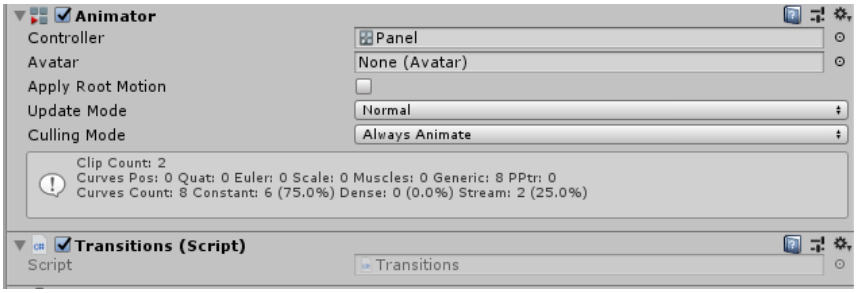
El primero es el de las transiciones en el cambio de escenas, donde se introduce entre medias un fundido a color negro para hacer más estético y disimulado la variación.

En primer lugar, se tienen que crear las animaciones de entrada y salida que van a representar la transición, contenidas en un Game Object Panel que va a hacerse cargo de controlar éstas.

Ambas animaciones juegan con la opacidad de una imagen de fondo negro, duran un segundo y se reproducen una detrás de la otra, sin embargo, su comportamiento es opuesto, pues la animación de entrada deberá modificar esa opacidad de nula a total y la animación de salida lo hará de total a nula.



Para insertar este detalle en las escenas simplemente se necesita un Game Object que tenga el componente *Animator* asociado y un script que se encargue de manejar este proceso.

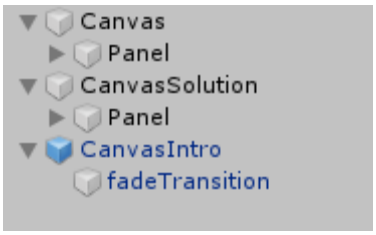


El script *Transitions* ejecuta corrutinas que se encargan de iniciar y finalizar las animaciones en función si se tiene que cargar una nueva escena o por el contrario ésta ya ha sido cargada.

```
0 references
public void loadScene() {
    StartCoroutine(startTransition());
}

1 reference
IEnumerator startTransition() {
    transitionAnim.SetTrigger("exit");
    yield return new WaitForSeconds(0);
}

1 reference
void sceneLoaded() {
    StartCoroutine(finishTransition());
}

1 reference
IEnumerator finishTransition() {
    yield return new WaitForSeconds(1);
    Destroy(fade);
}
```

El segundo patrón de ejecución que se repite tiene lugar en los tres minijuegos. En el concepto del transcurso de éstos existen tres partes claramente diferenciadas: la introducción (donde se presenta y explica el juego), el desarrollo del minijuego en sí y el final (donde se muestran las soluciones y los resultados una vez se ha completado el juego).

Estas tres partes también se encuentran separadas en la escena mediante el uso de diferentes Canvas, uno para cada sección, y es el script del minijuego el responsable de realizar los cambios entre estos durante la ejecución.



Para conseguir ese intercambio se ejecutan corrutinas de corta duración según una serie de propiedades que rige el minijuego donde se juega con la activación y desactivación de los diferentes objetos, con las siguientes instrucciones.

```
canvas.SetActive(true);    canvas.SetActive(false);
```

El tercer patrón de ejecución que se repite también se produce en los tres minijuegos, en concreto, una vez el usuario ha introducido la solución correcta del juego en cuestión. Visualmente, siempre se muestra el mismo panel, que corresponde con la pantalla de victoria y se puede ver en la siguiente imagen.

Por ese motivo, se ha creado un *prefab* llamado *winScreen* para que todos los minijuegos mantengan las mismas propiedades en cuanto a la estética de este objeto. La única diferencia que presentan y que ellos mismos se encargan de modificar es la variable del tiempo que el usuario emplea en completarlos.

Este aspecto tiene lugar en el momento de finalizar el minijuego, que se ejecuta de forma idéntica en todos dado que siempre se tienen que reproducir los mismos pasos, que son los siguientes:

1. Asignar el tiempo actual en la variable que corresponde a la pantalla de victoria.
2. Actualizar la información del Game Manager, almacenando el tiempo empleado por el usuario en ese minijuego e incrementar el minijuego actual al siguiente.
3. Reproducir el sonido de victoria.
4. Activar el Game Object de la pantalla de victoria para que salga en la escena.

```
void minigameCompleted() {
    Text finalTime = winScreen.GetComponentInChildren<Text>();
    finalTime.text = time.text;

    gameManager.TimeMinigames.Add(gameManager.ActualMinigame, t);
    gameManager.ActualMinigame++;

    sound.clip = Resources.Load<AudioClip>("Sounds/victoria");
    sound.Play();
    winScreen.gameObject.SetActive(true);
}
```

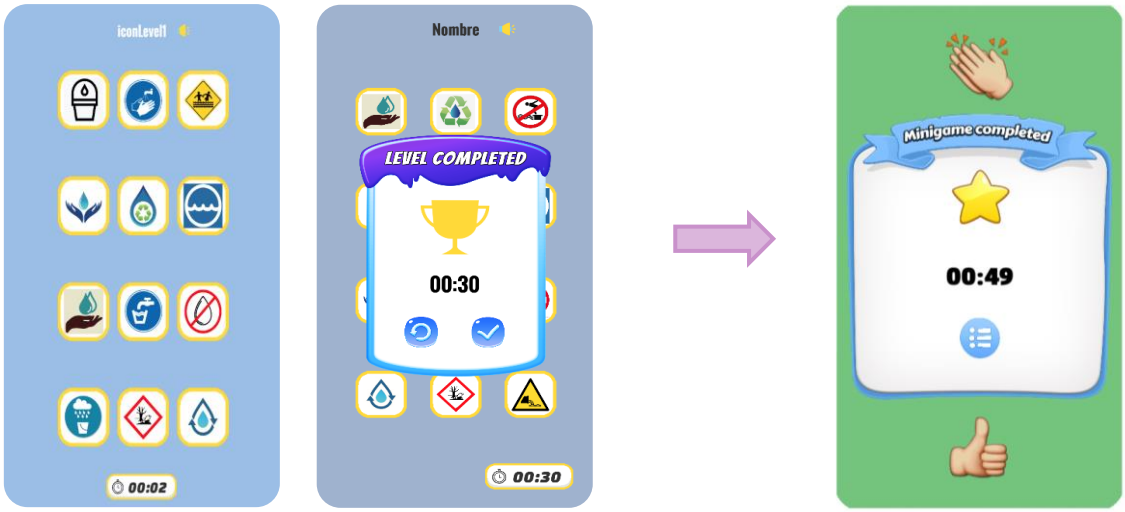
6.3 Cambios producidos

Durante el desarrollo se han realizado una variedad de cambios en la aplicación, tanto a nivel de código como a nivel de interfaz.

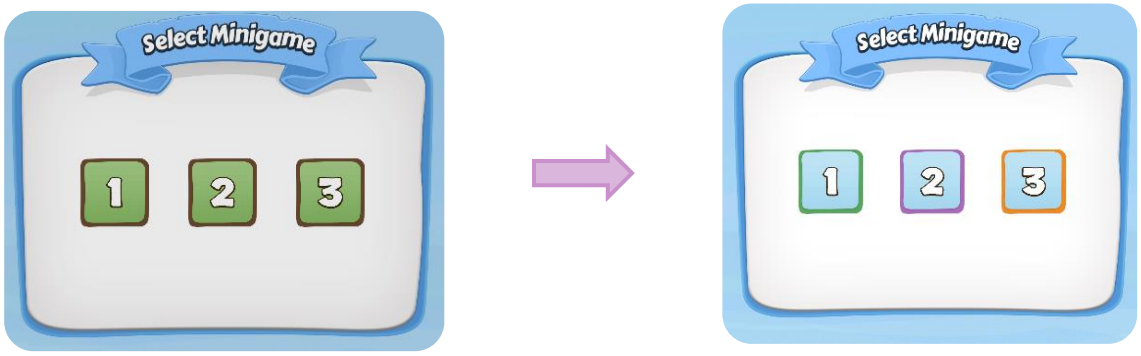
A nivel de código, las refactorizaciones realizadas han sido en busca de mejorar la eficiencia y rendimiento del videojuego, como por ejemplo que el algoritmo de mezcla de los iconos en el primer minijuego se realice en segundo plano mientras se muestran las soluciones y no al empezar el nivel, como se hacía en un principio.

Sin embargo, a nivel de interfaz sí ha habido más cambios y todos han ido enfocados a aumentar la sencillez y estética del juego, con componentes diseñados mucho más complejos y atractivos y editados para combinarlos con los colores del juego.

En primer lugar, se sustituyeron los recursos obtenidos de la *Asset Store* por otros de licencia gratuita mucho más personalizables y enfocados para teléfonos móviles (*Mobile Game GUI* [6]). En estos componentes se llevaron a cabo cambios en cuanto a colores, tamaño, fuentes y contenido con tal de adecuarlos al juego. Este hecho se puede detectar a simple vista comparando las tres imágenes que se muestran a continuación, especialmente en el panel de victoria.



Otro cambio que se produjo fue el de asignar un color a cada minijuego. Este hecho se ha reflejado tanto en el fondo de cada minijuego, que por defecto siempre era el mismo, como en el menú principal, donde los botones tenían el mismo diseño y color.



Ahora, por el contrario, se puede ver una relación en cuanto a colores y minijuegos e incluso ayuda al usuario a distinguirlos unos de otros.

7. Pruebas

En este apartado se van a detallar las pruebas que se han llevado a cabo en la aplicación. Se han realizado diferentes tipos de pruebas para evaluar la calidad del producto, garantizar el correcto funcionamiento de la aplicación y su validación por parte de los usuarios.

7.1 Pruebas unitarias

Se trata de un tipo de prueba funcional en la cual se busca comprobar el funcionamiento de una unidad de código concreta, ver cómo interactúa y localizar sus posibles errores.

Para esta cuestión se ha hecho uso de *Unity Test Runner* [12], una herramienta nativa del propio programa que integra la biblioteca *NUnit* y que se encarga de testear el código tanto en modo edición como en modo reproducción (*Play Mode*). En este caso, al tratarse de pruebas unitarias se ha empleado el modo de edición y se ha evaluado el componente del Game Manager, en concreto los métodos consultores y modificadores de sus variables.

Los resultados obtenidos se reflejan en la siguiente imagen.

7.1.1 Conclusiones

En la imagen anterior se puede ver que todas las pruebas realizadas han resultado satisfactorias y no se ha producido ningún error.

7.2 Pruebas de compatibilidad

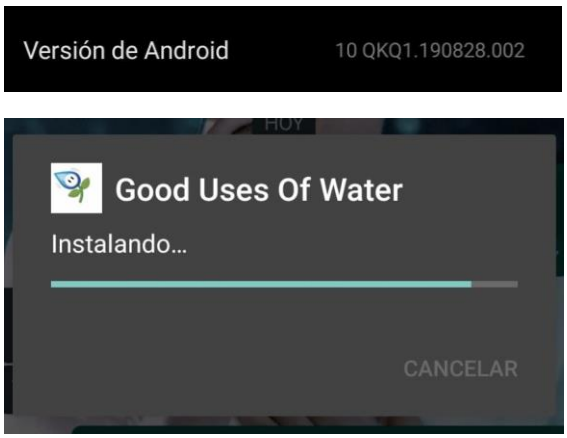
Se trata de un tipo de prueba no funcional en la cual se busca verificar el funcionamiento del sistema en diferentes entornos. Este tipo de testing se puede realizar de diversas formas, pues se puede probar en diferentes navegadores, sistemas operativos, dispositivos…

En este proyecto se ha realizado probando la aplicación en diferentes versiones del sistema operativo Android: Android Jelly Bean (API 16), Android Oreo (API 26) y Android Q (API 29) y en dispositivos con diferentes resoluciones (tablet de 10.1 pulgadas y móvil de 6,39 pulgadas).

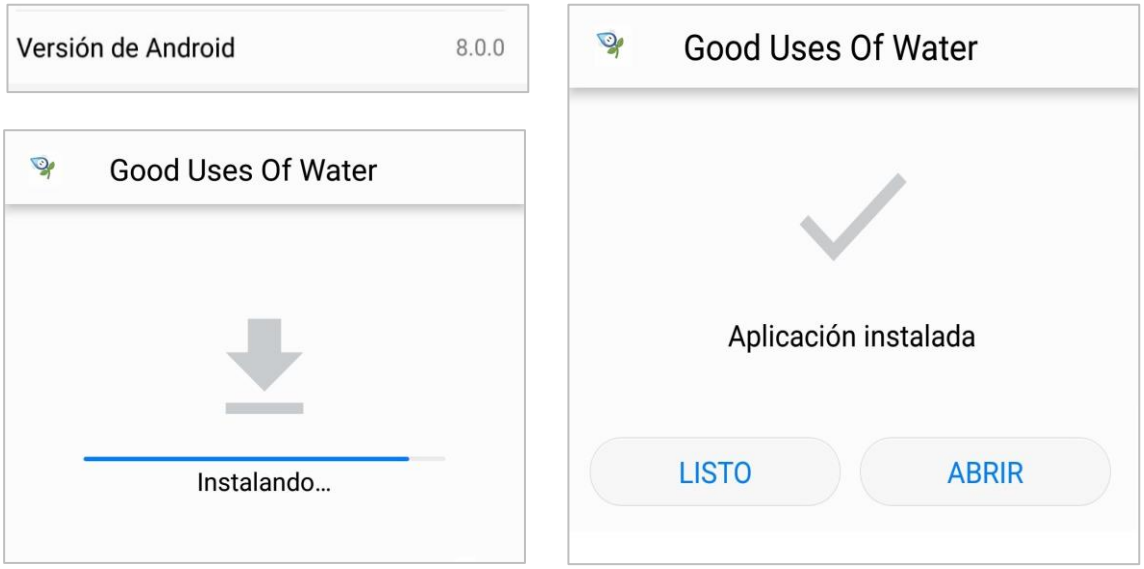
Debido a que esta aplicación exige como mínimo una versión de Android 4.4 (API 19) se han escogido las versiones mencionadas anteriormente, donde se busca comprobar que se sigue el comportamiento esperado en torno a este requisito.

En la siguiente imagen se muestra la especificación de este hecho en la aplicación.

7.2.1 Prueba 1: Android 10 (API 29)



7.2.2 Prueba 2: Android 8 (API 26)



7.2.3 Prueba 3: Android 4.1 (API 16)

7.2.4 Conclusiones

Tal y como se puede observar en las capturas de los apartados anteriores, se permite la instalación de la aplicación en las versiones de API 29 y API 26, y, por el contrario, en la versión API 16 no es posible su instalación.

Además, en los dos dispositivos en los que se ha ejecutado se ha podido comprobar que la interfaz cumple el requisito de ser *responsive,* adaptándose a las diferentes resoluciones y mostrarse uniforme.

Por lo tanto, se puede concluir que los resultados de todas las pruebas realizadas han sido satisfactorios.

7.3 Pruebas de usabilidad

Se trata de un tipo de prueba no funcional en la cual se busca evaluar el producto mediante pruebas con usurarios. Se enfoca en medir la capacidad de éste en cumplir el propósito y alcance para el que fue diseñado.

En este proyecto se ha realizado probando la aplicación con tres usuarios diferentes cuyo rango de edad discurre entre los cinco y trece años.

7.3.1 Usuario 1: 5 años.

PANTALLA	OBSERVACIONES
Pantalla de bienvenida (iniciar juego)	El usuario es capaz de iniciar el juego sin problemas.
Menú principal (iniciar minijuego correspondiente)	El usuario comprende a qué niveles puede acceder e inicia el minijuego sin problemas.
Minijuego 1	El usuario emplea 03:33 en completar el minijuego y le cuesta comprender algunos niveles.
Minijuego 2	El usuario emplea 02:51 en completar el minijuego y necesita explicación para jugar.
Minijuego 3	El usuario emplea 01:13 en completar el minijuego.
Menú final (resultados y salir del juego)	El usuario entiende los resultados que se muestran y es capaz de salir del juego sin problemas.

7.3.2 Usuario 2: 8 años

PANTALLA	OBSERVACIONES
Pantalla de bienvenida (iniciar juego)	El usuario es capaz de iniciar el juego sin problemas.
Menú principal (iniciar minijuego correspondiente)	El usuario comprende a qué niveles puede acceder e inicia el minijuego sin problemas.
Minijuego 1	El usuario emplea 03:21 en completar el minijuego y le cuesta encontrar la solución del nivel 4.
Minijuego 2	El usuario emplea 01:41 en completar el minijuego y duda del funcionamiento del juego.
Minijuego 3	El usuario emplea 01:04 en completar el minijuego.
Menú final (resultados y salir del juego)	El usuario entiende los resultados que se muestran y es capaz de salir del juego sin problemas.

7.3.3 Usuario 3: 13 años

PANTALLA	OBSERVACIONES
Pantalla de bienvenida (iniciar juego)	El usuario es capaz de iniciar el juego sin problemas.
Menú principal (iniciar minijuego correspondiente)	El usuario comprende a qué niveles puede acceder e inicia el minijuego sin problemas.

Minijuego 1	El usuario emplea 01:41 en completar el minijuego y le cuesta encontrar la solución del nivel 4.
Minijuego 2	El usuario emplea 02:27 en completar el minijuego.
Minijuego 3	El usuario emplea 00:55 en completar el minijuego.
Menú final (resultados y salir del juego)	El usuario entiende los resultados que se muestran y es capaz de salir del juego sin problemas.

7.3.4 Conclusiones

Tal y como se puede observar en las tablas anteriores el juego es lo suficientemente entendible e intuitivo por sí mismo como para que los usuarios puedan ejecutarlo de principio a fin sin problemas.

En general, el minijuego que lleva más tiempo completar es el primero (tal vez por ser el más largo) y el minijuego que se finaliza más rápido es el último. Además, todos los usuarios coinciden en que éste es el que más les gusta por ser más divertido y ameno, les parece entretenido el tener que arrastrar los iconos al círculo en el minijuego 2, aunque se agradecería una explicación más concreta, y agradecen que no sea repetitivo, evitando el funcionamiento monótono.

Por otra parte, también se puede confirmar que cuánto más pequeño es el usuario, más tiempo emplean en completar los minijuegos.

Por lo tanto, se puede concluir que los resultados de todas las pruebas realizadas han sido satisfactorios.

8. Conclusión

En este último apartado se va a resumir el conjunto de conclusiones a las que se ha llegado una vez finalizado el proyecto. Además, se van a sugerir una serie de propuestas como mejoras de la aplicación que podrían ser incluidas en un futuro. Por último, se realizará una valoración personal del trabajo realizado.

8.1 Conclusiones

Con respecto al alcance de la aplicación propuesto en un principio, se puede confirmar que se satisfacen todas las expectativas y todos los requisitos, funcionales y no funcionales, que el cliente exigía.

Tras las pruebas de usabilidad con los usuarios también se puede decir que se cumplen todos los objetivos del proyecto y que en general el resultado de éste ha sido bastante exitoso.

8.2 Propuestas de mejora

Una vez finalizada la aplicación y testeada surgen varias posibles mejoras que, además de hacerla más completa, ayudaría mucho más al usuario para su uso y a la vez atraería más su atención.

- **Pantallas de introducción más explicativas.** Una de las conclusiones a las que se llega tras las pruebas es que el segundo minijuego necesita de una explicación previa en los usuarios de corta edad. Con un texto más descriptivo puede que esa cuestión estuviera solucionada.

- **Vídeos de ejemplo para mostrar el funcionamiento del juego.** Otra opción para solucionar el problema anterior y además hacer la aplicación más interactiva es la de mostrar un video explicativo antes del minijuego, con el fin de explicar visualmente cómo se ha de jugar. Hay que destacar que una de sus ventajas es que los usuarios no necesitarían leer, lo que supone una clara ayuda para el sector analfabeto del que hemos hablado al principio del documento.

- **Posibilidad de guardar la partida.** Añadir esta funcionalidad sería un gran avance, pues el usuario no estaría obligado a completar el juego de principio a fin sin pausas; podría guardar el estado del progreso que lleva actualmente en el juego y continuar más adelante desde ese mismo punto.

- **Añadir más minijuegos.** Esta mejora requiere un mantenimiento de la aplicación de aquí en adelante, pero es la que garantiza el uso continuado de la aplicación y que no se quede estancada una vez completado el juego y, al mismo tiempo, la que proporciona novedad.

8.3 Valoración personal y relación con los estudios cursados

Los cuatro años de grado cursados y los conocimientos adquiridos en ellos han supuesto de base y de gran utilidad para el proyecto actual. Por lo tanto, las asignaturas que tienen relación directa con el proyecto son:

- Estructura de datos.
- Programación.
- Bases de datos y sistemas de información.
- Interfaces persona computador.
- Ingeniería del software.

Y especialmente, aquellas que se han cursado durante los dos últimos años, en la rama de Ingeniería del Software y que son más específicas:

- Gestión de proyectos.
- Diseño de software.
- Calidad del software.
- Proceso de software.
- Mantenimiento y evolución del software.
- Análisis y especificación de requisitos.
- Proyecto de ingeniería de software.

Con respecto al desarrollo de videojuegos hay que destacar que con ya había llevado a cabo un proyecto en la asignatura de Proceso del software que requería el uso del programa Unity y del cual se realizó una formación externa. Por lo tanto, había un dominio previo de la tecnología empleada, pese a que no formaba parte de la teoría cursada.

No obstante, el mayor problema se ha producido en este aspecto, pues era la primera vez que se desarrollaba un videojuego para dispositivos móviles y la cuestión de la interfaz es la que ha ocasionado más contratiempos.

La dificultad se encontraba concretamente en conseguir la uniformidad y adaptabilidad de la aplicación en diferentes resoluciones, uno de los requisitos más importantes.

Así que en relación con este campo se puede afirmar que el dominio de la herramienta Unity y el conocimiento acerca de la programación de videojuegos en sí se ha visto afianzado mucho más.

Otro aspecto que también ha servido de ayuda es el de haber formado parte con anterioridad de varios proyectos que partían desde cero, es decir, desde la fase de especificación con el cliente hasta la fase de despliegue, con una metodología de trabajo ágil. Estos proyectos se han desarrollado tanto dentro de la universidad como fuera, de forma individual, en grupo y en prácticas de empresa, de modo que ya me sentía familiarizada con esa forma de trabajo.

No obstante, este proyecto me ha servido para mejorar como profesional en este campo, aplicando los conocimientos adquiridos de forma individual en un contexto real y la experiencia ha sido muy satisfactoria.

9. Referencias

[1] Abad Cerdá, F.J. (Curso académico 2019-2020). *Apuntes de Entornos de desarrollo de videojuegos*.

[2] Andrade, S. (25 de abril del 2015). *Unity: Controlador táctil para dispositivos móviles (Nuevo Sistema UI)*. [Última consulta: 13 de febrero de 2020] Disponible en: http://aev.org.es/unity-controlador-tactil-para-dispositivos-moviles-nuevo-sistema-ui.

[3] Apple (12 de agosto del 2015). *Nature Bert Save The Earth.* [Última consulta: 15 de abril del 2020]. Obtenido de: https://apps.apple.com/us/app/nature-bert-save-the-earth-juegos-educativos-para-ni%C3%B1os/id892086298?l=es

[4] Blanco Bueno, C. (2019). *Apuntes de desarrollo de videojuegos: Game Manager.* [Última consulta: 24 de enero del 2020]. Disponible en: https://personales.unican.es/blancobc/apuntesvj/prog/gameManager.html.

[5] European Union Apps (07 de diciembre del 2012). *El increíble chapuzón de Zoe*. [Última consulta: 15 de abril del 2020]. Obtenido de la Comisión Europea: https://ec.europa.eu/environment/pubs/children/zoe/index_es.htm.

[6] GraphicBurger (16 de septiembre de 2013). *Mobile Game GUI.* [Fecha de consulta: 12 de febrero del 2020]. Disponible en: https://graphicburger.com/mobile-game-gui/.

[7] Lidon Mañas, M (31 de diciembre del 2018) *Unity 3D*. Marcombo. ISBN 9786075384603.

[8] Marketing directo (29 de enero del 2012). *Los tables reemplazan al bolígrafo y al papel en las escuelas indias.* [Última consulta: Última consulta: 10 de abril 2020]. Obtenido de: https://www.marketingdirecto.com/digital-general/mobile-marketing/los-tablets-reemplazan-al-boligrafo-y-al-papel-en-las-escuelas-indias.

[9] Microsoft (2017). *Guía de C#*. [Última consulta: 12 de mayo del 2020] Disponible en: https://docs.microsoft.com/es-es/dotnet/csharp.

[10] Knowre (2019). *Knowre Math.* [Última consulta: 15 de abril 2020] Disponible en: https://www.knowre.com/knowre-math.

[11] Unity Technologies (2010). *Asset Store.* [Última consulta: 22 de abril 2020] Disponible en: https://assetstore.unity.com/.

[12] Unity Technologies (2019). *Documentation: Unity User Manual.* [Última consulta: 06 de junio 2020] Disponible en: https://docs.unity3d.com/Manual/UnityManual.html.

[13] Wikipedia (04 de noviembre del 2005). *CSharp.* [Última consulta: 17 de mayo 2020] Disponible en: https://es.wikipedia.org/wiki/C_Sharp.

[14] Wikipedia (14 de febrero del 2018). *Visual Studio Code.* [Última consulta: 17 de mayo 2020] Disponible en: https://es.wikipedia.org/wiki/Visual_Studio_Code